E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Juego serio en Realidad Virtual para dispositivos Android y Oculus, mediante el empleo de Unity, Arduino y Raspberry Pi

Grado en Ingeniería Informática

Trabajo Fin de Grado

Unai Domínguez Beltrán

Oscar Ardaiz Villanueva

Pamplona, junio de 2019

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ÍNDICE DEL PROYECTO

1. PRESENTACIÓN E INTRODUCCIÓN AL PROYECTO "PULSACIONES VR"

1.1 ¿Por qué escogí esta idea?

Antes de empezar a describir el proyecto que he implementado como trabajo final del Grado en Ingeniería Informática me gustaría hacer una pequeña presentación sobre mi persona, con el propósito de entender el porqué de mi decisión de elegir la temática escogida.

Desde muy pequeño me he sentido muy atraído por el mundo de los videojuegos y todo lo que le rodea, no solo por la diversión que ofrecen, sino también por lo que puede aprenderse de ellos.

Aún recuerdo cuando me regalaron con 6 años el título "*The Legend of Zelda – Links Awakening*", el cual no estaba traducido al castellano. Por aquel entonces en Internet no había la gran cantidad de ayudas o traducciones que ahora si pueden verse y te tocaba "sacarte las castañas del fuego" como pudieses. Quizás tardabas mucho más tiempo en terminar los juegos que las nuevas generaciones por ese motivo, sin embargo, sin buscarlo, aprendías más inglés que quizás en varios cursos en el colegio.

A eso hay que añadir la gran cantidad de valores que uno aprende, aunque claro, excepciones hay en todos los sitios.

No solo eso. Muchas veces he escuchado la frase "deja la máquina y sal a la calle a jugar a cualquier otra cosa". Quizás en mi caso y en el de mucha gente sea aplicable, pero no siempre es así. Personas con algún tipo de discapacidad motora pueden encontrar en este mundo una vía para poder disfrutar de situaciones en las que en la vida real les resulta muy difícil.

Además, últimamente está en auge la idea de juegos en realidad virtual mediante los cuales la inversión en los mismos, dependiendo también la plataforma que se utilice, puede ser total, haciendo creer a la persona que está disfrutando del mismo que en realidad está dentro de éste.

Por lo que significó en la infancia para mí, uno de mis mayores sueños es el poder transmitir todo lo que este universo me dio y me sigue dando hoy en día a la mayor cantidad de público posible, ya sean jóvenes o no. El camino es bastante difícil, lo reconozco, pero ¿quién dijo que no lo fuera?

Para empezar con este trayecto decidí entrar en la carrera de Ingeniería Informática con el objetivo de aprender todo lo posible sobre el mundo de la programación. Sabía que lo más probable es que no hubiera asignaturas dedicadas al desarrollo de videojuegos, pero que sí me darían una base para poder desenvolverme ante cualquier tipo de lenguaje que necesitara en un futuro.

Es cierto que podía haber escogido cualquier grado dedicado a este mundo, pero buscando información muchas personas te recomendaban que primero se empezara con este grado para después especializarte con un Máster de este tipo.

Pese a que a lo largo de la carrera esta idea se fuese diluyendo debido a nuevas temáticas que te iban presentando, en tercero tuve la suerte de poder disfrutar la asignatura de "Sistemas Multimedia y diseño centrado en el usuario" que en el momento que la cursé estaba orientada al desarrollo de videojuegos mediante **Unity**. Aunque ya había trasteado con él (nada serio), fue en ese instante en el que supe que no debía desviarme del planteamiento con el que entre en el grado. Era muy gratificante ver las caras de ilusión de las personas a las que les ibas mostrando los diferentes proyectos, que solo con eso ya merecía todo el esfuerzo y sudor que costaba sacarlos adelante.

Tenía claro que para mi trabajo fin de grado quería desarrollar aplicar todo lo aprendido en esa clase, y si pudiera aprender nuevos conocimientos, mucho mejor.

Fue entonces cuando a principios de septiembre, el profesor de la universidad Oscar Ardaiz Villanueva nos mandó un mensaje a todos los alumnos del grado que empezábamos el último curso con una serie de propuestas para este proyecto. Entre ellas, llamó mucho mi atención un proyecto en el que había que desarrollar un juego serio para realidad virtual mediante Unity. Sabía que se habían alineado los astros y no debía dejar escapar la oportunidad.

En los últimos años el tema de la realidad virtual ha cogido bastante fuerza con proyectos como **Oculus Rift**. La idea de poder sumergirte en todos aquellos mundos de fantasía que imaginabas en tu infancia y poder a llegar a vivirlas como si fueran ciertas fue otro de los motivos de la elección de este proyecto.

1.2 Introducción al juego "Pulsaciones VR"

Tras un largo y duro trabajo varios meses he podido completar el desarrollo del juego Pulsaciones VR, en el cual trato de trasladar una habitación real a un entorno virtual en el cual le doto de una cierta funcionalidad.

En concreto, la idea es establecer un sistema de pulsadores en dos de las paredes de la sala, como puede apreciarse en las siguientes imágenes (más tarde entraremos a explicar en profundidad como funciona):

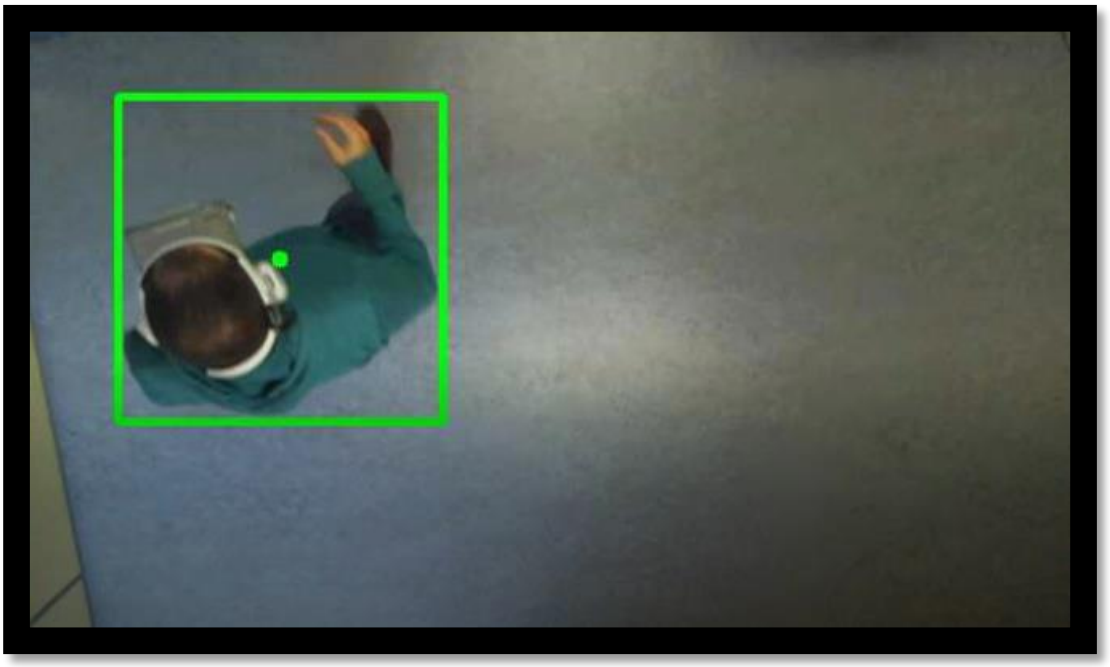
1 Disposición de los pulsadores en el escenario real

De los seis posibles pulsadores, solo aparecerá cada vez un pulsador activo, que debemos presionar para avanzar en el juego. Al pulsarlo este desaparece y otro es activado de manera aleatoria.

El juego está formado por 4 niveles y en cada nivel debes de presionar un número determinado de veces. Más adelante explicaremos las características de cada uno de los niveles.

Cada trío de pulsadores se encuentra conectado a una placa Arduino y cada vez que uno de éstos es presionado, manda una señal de tipo OSC al juego. Al recibirla, si has presionado el pulsador correcto éste desaparecerá activando otro pulsador.

Otra cuestión para analizar es como conseguimos movernos a la vez en el juego y en la vida real. Para llevarlo a cabo tenemos situada en el techo una cámara conectada a una Raspberry Pi 3 B. Esta se encarga de calcular la posición del jugador dentro de la sala y enviarla al juego también por medio de mensajes tipo OSC.



2 Reconocimiento de la posición del jugador

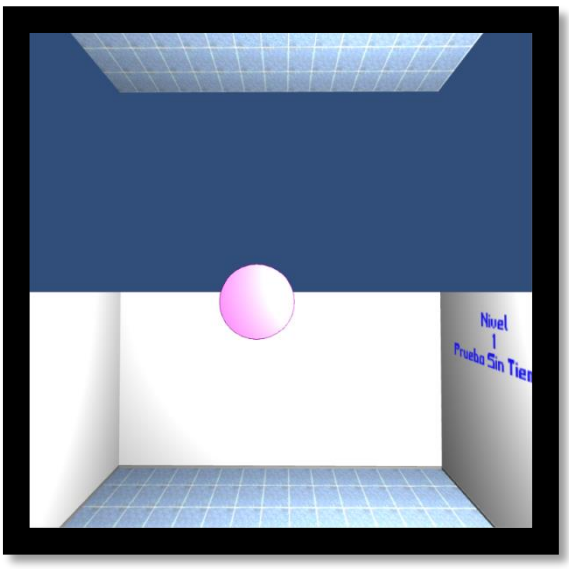
2. EXPLICACIÓN DE LOS NIVELES DEL JUEGO

Como he mencionado en el apartado anterior, el juego dispone de 4 niveles que el jugador debe superar. Los dos últimos tienen un temporizador de un minuto y en el caso de no poder presionar todos los pulsadores que son solicitados, el jugador perderá la partida. Para que éste pueda adaptarse al entorno, los dos primeros niveles son sin temporizador.

En cada uno de los niveles se han introducido diferentes indicaciones tanto visuales como auditivas. Cada uno de los pulsadores tiene asociado un sonido (es el mismo para todos, solo que cada uno tiene asociada una frecuencia diferente), que suena en el momento que alguno de ellos se activa. Por su parte también se indica al jugador si se ha producido un cambio de nivel.

2.1 Primer Nivel

Cuando empieza el juego te encuentras con el siguiente escenario:

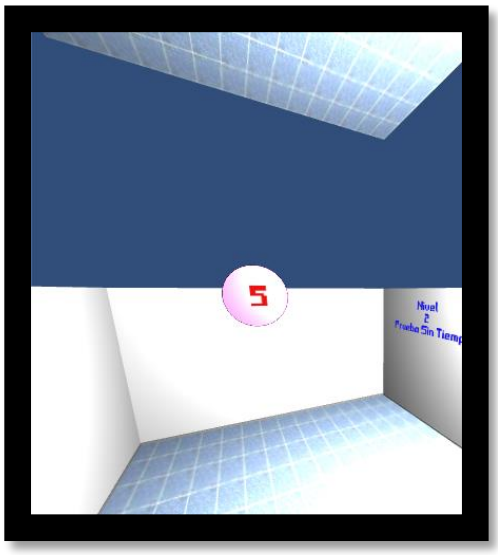


Como he mencionado previamente, tenemos indicaciones a nivel auditivo y visual. Con respecto a este último caso, en una de las paredes de la habitación aparecerá información asociada a la situación del nivel. En concreto, se nos informa sobre el nivel en el que nos encontramos, el tiempo que tenemos para terminar el mismo, y el número de pulsaciones que necesitamos realizar.

Como también ocurre en el nivel siguiente, al tratarse de escenarios de aprendizaje para que la persona se acostumbre a desplazarse por el entorno, el temporizador estará desactivado (se puede apreciar esta situación tanto en el bloque de texto vinculado al nivel, como al del cronómetro).

A medida que vayamos desactivando los pulsadores, el contador de pulsaciones irá disminuyendo:

2.2 Segundo Nivel



Aunque también se trate de un nivel de adaptación, éste segundo escenario tiene bastante más dificultad, como consecuencia de haber añadido un cronometro asociado a cada uno de los pulsadores.

Si el contador llega a 0 desaparecerá y se activará aleatoriamente otro de los restantes sin producirse ningún cambio en las pulsaciones que quedan para avanzar de nivel.

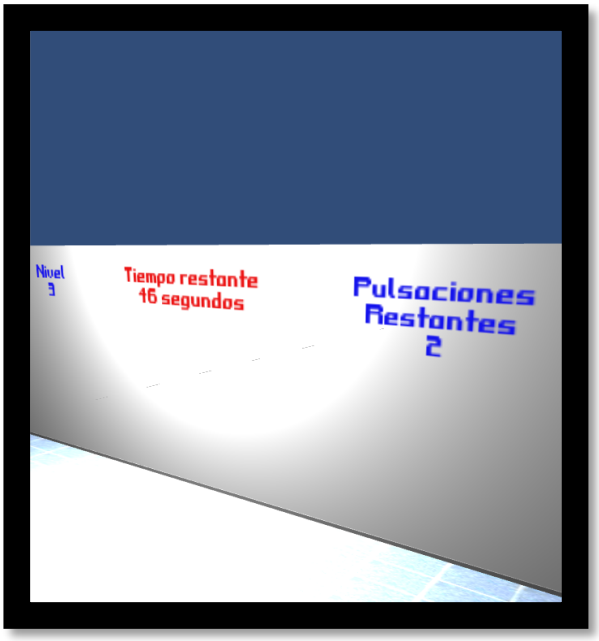
Como puede observarse en la imagen de la izquierda, el valor del cronómetro será visible en todo momento, por lo que el usuario del juego siempre tiene a su disposición el tiempo que le queda para pulsar el botón.

Con respecto a la información que es mostrada en la pared no se produce ningún cambio significativo:

2.3 Tercer Nivel

El funcionamiento del tercer nivel es similar al visto en el primero, con la salvedad de que en este caso no podemos relajarnos, ya que existe un cronometro vinculado al mismo que en caso de llegar a 0 hará que perdamos la partida.

Al iniciar el nivel podemos contemplar que el contador no varía. Esto es debido a que hasta el momento en que no presiones ninguno de los botones no empezará la cuenta atrás.



Otro de los cambios significativos que ocurren en este nivel como en el siguiente es que aparecen más indicaciones auditivas las cuales nos van informando sobre el tiempo restante:

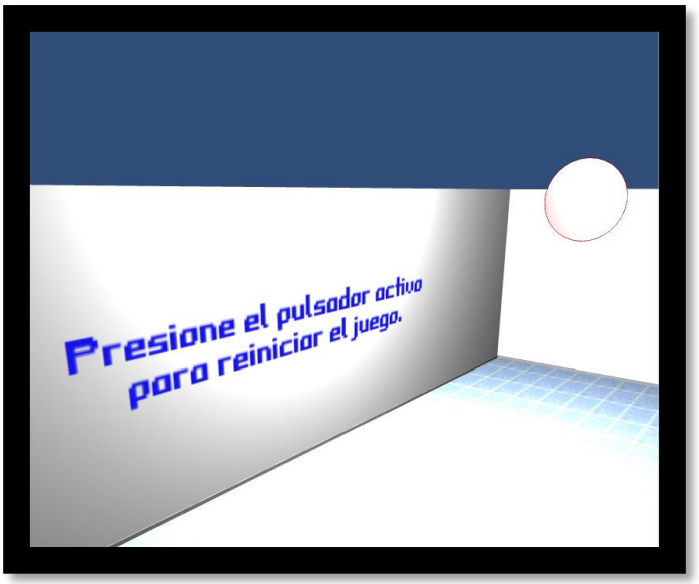
Al llegar a la mitad del tiempo restante, se oirá: "***Faltan 30 segundos***"

Cuando únicamente queden 10 segundos, se escuchará: "**Solo 10 segundos, rápido**"

En caso de que el cronómetro llegue a su fin se activará el siguiente mensaje: "***Tiempo***"

Si se da este último caso, se le ofrecerá la opción al jugador de reiniciar el juego. Para realizarlo, simplemente debe pulsar el pulsador que esté activado en ese instante. Esto es mencionado en la pared de información y mediante otro comentario que dice lo siguiente:

"***Presione el pulsador visible para volver a activar el juego***"



2.4 Nivel Final

Como ocurre en el caso anterior, este nivel hace referencia a uno de los dos niveles que hemos superado al principio sin temporizador, en concreto al segundo. Con respecto a las indicaciones es muy similar el tercer nivel, sin embargo, hay que incluir un añadido. En el supuesto caso en el que el jugador logre completar el nivel, se escuchará un mensaje felicitándole por completar el juego, el cual dice lo siguiente:

"***Felicidades, has completado todos los niveles del juego***"

Posteriormente se le ofrecerá al jugador la opción de reiniciar el programa de la misma forma que ocurría en el caso de perder.

3. ESTUDIO DEL ARTE

Como he mencionado en el primer punto de este documento el mundo de la realidad virtual está en auge, por lo que en el mercado podemos encontrarnos una gran cantidad de juegos que implementan este sistema.

Con respecto a la idea de proyecto que he desarrollado para este trabajo no he encontrado nada que mezcle la detección de posicionamiento de una persona gracias a una Raspberry Pi para trasladarla al juego junto con el sistema de "*Capacitive Touch*" de Arduino.

Por separado, de esta última idea sí que hay una gran cantidad de proyectos asociados, desde pianos hasta juegos como "**Simón dice**":

a) Piano (Proyecto desarrollado por Tijmen van Willigen) [1]

b) Simón Dice (Proyecto desarrollado por Abdualziz Alawshan) [2]

Sin embargo, para el tema de detección de posición de una persona mediante una Raspberry Pi y una cámara conectada a esta no he encontrado ningún juego que lleve a cabo esta idea. Sí que podemos encontrar sistemas desarrollados para la detección de movimiento con esta estructura, por ejemplo, para la construcción de una cámara de seguridad:

c) Proyecto desarrollado por Hacker Shack [3]

d) Proyecto similar a la aplicación Pulsaciones VR – VirtualSpace [4]

A pesar de que el modo de funcionamiento es diferente, sí que este proyecto desarrollado por el Instituto **Hasso Plattner** guarda cierta relación con la idea que he tratado de trasladar en mi proyecto **Pulsaciones VR,** de emplear una habitación como zona de desplazamiento, y que este hecho afecte también al movimiento del usuario del juego.

Al contrario que en el trabajo que he realizado, en dónde la detección del jugador es calculada por cada uno de los *frames* que la cámara va realizando, emplea un sistema de láseres para detectar los posicionamientos de las personas que se encuentran en la sala.

Teniendo en cuenta la detección de la posición del jugador por láseres visto en el proyecto de **VirtualSpace,** si nos vamos a un nivel superior de tecnología, como **Oculus Rift**, sí que podemos encontrar juegos de realidad virtual en los que podemos desplazarnos libremente por una habitación siendo este desplazamiento captado por el juego, sin el empleo del sistema de Raspberry.

Para conseguir esto puede vincularse a este dispositivo hasta cuatro sensores de movimiento propios del sistema (**Constellation**). El funcionamiento de este hardware es sencillo. Tanto en las gafas del Oculus Rift como en sus mandos (los llamados **Oculus Touch**) disponen de una gran cantidad de marcadores organizados como una constelación (de ahí el nombre del sistema) los cuales son captados por los sensores cuadro por cuadro. Más tarde esta información es procesada por el software de Oculus para conocer la posición exacta del usuario dentro de un área previamente delimitada a la hora de realizar la configuración del sistema. Previamente es necesario delimitar la zona en la cual nos desplazaremos [5].

Podemos ver el funcionamiento de las Oculus Rift y el sensor de movimiento en el siguiente video de YouTube en donde su propietario juega a la demo "**Oculus First Contact**" [6].

En el departamento donde he implementado el juego disponemos de este sistema. Por lo que, para ver la diferencia entre los diferentes métodos de desarrollo exporté el juego al mismo. El único cambio por realizar era el de no calcular la posición del usuario mediante la comparación de fotogramas, sino emplear el procedimiento descrito en el párrafo anterior.

4. HERRAMIENTAS UTILIZADAS

4.1 A nivel de hardware

4.1.1 Raspberry Pi 3 (Modelo B)

En los últimos años ha cobrado especial importancia el concepto de Raspberry Pi, pero ¿en qué consiste exactamente y por qué lo he empleado en el desarrollo de mi proyecto?

Se trata de uno de los microordenadores más económicos y rentables del mercado y como consecuencia de esto, la explotación de este hardware ha llegado a límites insospechados. Con él puedes diseñar multitud de proyectos de todo tipo, desde recreativas (mediante los sistemas operativos **Recalbox** o **Retropie** ambos basados en Linux) y centros multimedia (**KODI**) hasta sistemas de estaciones meteorológicas precisas.

Las características de este sistema han variado mucho desde que salió a la venta su primera versión. A pesar de que existen en el mercado versiones más actuales del modelo 3B (en concreto 3B+), sus características no le tienen nada que envidiar.

Dispone de:

- **Procesador**

  Chipset Broadcom BCM2387 (1,2 GHz de cuatro núcleos ARM)

- **1GB de Memoria RAM**
- **Ranura para tarjeta de memoria MicroSD**
- **Conectividad:**
  - Ethernet socket Ethernet 10/100 BaseT
  - Wifi
  - Bluetooth 4.1
  - Salida de audio/video HDMI y RCA compuesto
  - Conector USB 2.0 (en concreto 4)

¿Cuál es el uso pensado para este "microordenador"?

La idea es instalar un sistema operativo en él donde poder gestionar el posicionamiento de la persona que está utilizando el juego. En concreto me decidí por instalar el sistema **Raspbian Jessie** junto con las librerías de Python de OpenCV3 para conseguir este objetivo [7][8].

Para poder visualizar lo que la Raspberry Pi estaba mostrando podíamos hacerlo de tres maneras diferentes, mediante conexión HDMI a un monitor, empleando adaptadores HDMI-VGA o mediante VNC (Virtual Network Computing).

Nos decantamos por la última opción como consecuencia de que en el primer caso las pantallas del departamento que disponían de conexión HDMI no reconocían el sistema y en el segundo, en las que solo disponían de conexión VGA no logramos obtener una resolución óptima.

Por su parte, la tecnología VNC nos permitía transmitir de manera remota a través de red el escritorio de un equipo a otro en tiempo real. A pesar de que en función del tipo de pantalla que se esté empleando para la visualización del escritorio de la Raspberry fuera necesario ajustar las resoluciones, este no fue mi caso, ni cuando realizaba la conexión con mi portátil, como cuando empleaba el ordenador del laboratorio para este fin.

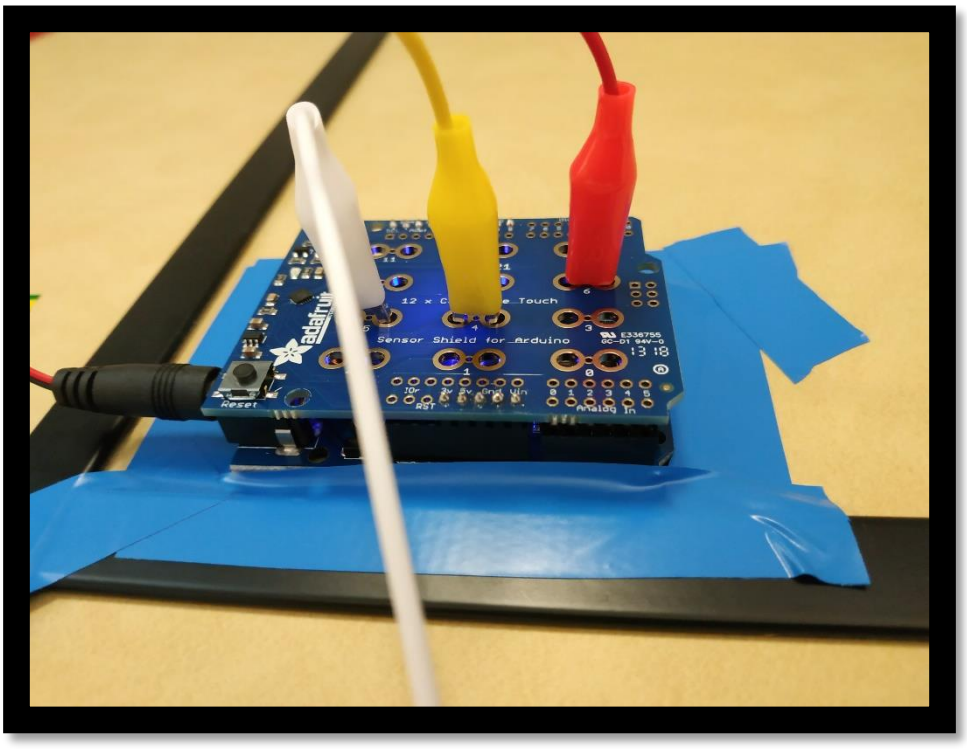
Para captar información de la sala (grabación en tiempo real), utilizamos una cámara con conexión USB estándar.

¿Por qué no empleamos la cámara oficial de Raspberry (Pi Camera)?

Básicamente por el hecho de la altura a la que teníamos que poner la cámara en el techo. Esta cámara no se conecta mediante una conexión USB (se conecta directamente mediante un slot para cables planos) y no podíamos emplear ningún tipo de alargador para tener el microordenador conectado a la corriente a una altura accesible y solamente la cámara en la parte superior de la sala.

Observación: El tema del cálculo del posicionamiento del usuario será explicado al analizar el código en Python que se encarga de ello. Por ahora en este punto únicamente nos centraremos en la explicación del hardware y software utilizado.

4.1.2 Placa Arduino (WeMos D1 ESP8266 WIFI)



Como ocurre en el caso de Raspberry Pi, el concepto de Arduino también ha conseguido hacerse un gran hueco en el mundo de los proyectos software. A diferencia del primero que solo tiene una única versión que es renovada cada cierto tiempo, Arduino no cuenta con un único modelo. Esto ocurre ya que los fabricantes pueden crear sus propias placas de este tipo para satisfacer determinadas necesidades, siempre teniendo en cuenta determinadas características de hardware previamente definidas.

De la gran multitud de placas nos decantamos por la versión **WeMos D1 ESP8266,** entre otras cosas, por la conectividad WIFI que esta ofrece y porque se programa empleando el IDE de Arduino tanto en lenguaje C como C++. [9]

¿Cuál es el uso pensado para esta placa?

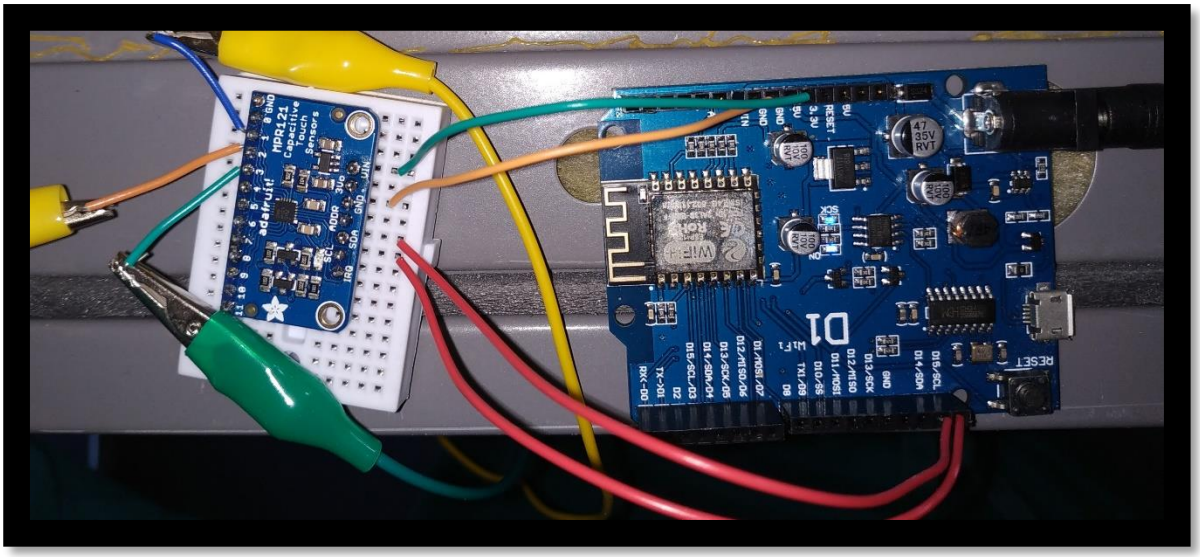
Necesitábamos algo que se encargara de reconocer el momento en que el jugador realiza los contactos con los diversos pulsadores y que se encargara de gestionar el envío de esta información al juego.

Empleando el "*Capacitive Touch Shield*" conectada a la placa y a los pulsadores (los cuales estaban realizados con adhesivos de aluminio) se podía conseguir este objetivo.

Dicho accesorio cuenta con 12 sensores capacitivos a los cuales se les pueden conectar pinzas de cocodrilo. Si en el lado que queda libre de estos cables es conectado algo que conduzca la electricidad (por ejemplo, el aluminio que hemos empleado en los pulsadores) y éste es presionado, el sistema detecta que uno de los electrodos de los sensores ha recibido información sobre este hecho.

A su vez, al tener que trabajar con dos placas Arduino para tener la estructura de pulsadores en dos de las paredes de la habitación, también decidimos emplear otro tipo de sistema capacitivo para no quedarnos solo con lo visto hasta ahora, el **MPR121 "***Capacitive Touch Sensors"* el cual sería conectado a una b*readboard* compatible con Arduino [10].

Observación: Al no contar con una *breadboard* con los rieles vinculados al suministro de energía eléctrica del circuito cuando una batería es conectada, las conexiones de cada uno de los pines de la placa deben conectarse directamente a las ranuras del Arduino. La estructura quedaría de la siguiente manera:



Pines utilizados:

- **Vin**

  Pin de la placa relacionado con la alimentación de esta. Utilizando un voltaje de 3.3V es suficiente para conseguirlo.

- **GND**

  Pin asociado a la toma de tierra. Aunque no hay ninguna batería conectada a la *breadboard,* el Arduino si que dispone de una de estas, por lo que es necesario utilizar este pin.

- **SCL y SDA**

  Pines relacionados con la comunicación entre las dos placas (bus de comunicación que emplea el protocolo síncrono **I2C**). El SDA manda datos por cada ciclo de reloj que le indica el SCL.

4.1.3 Dispositivos móviles

He querido realizar una puntualización en esta sección sobre el tema de los dispositivos móviles en los que se puede instalar esta aplicación. Únicamente funciona en aquellos dispositivos Android cuya versión sea como mínimo **4.4 'KitKat'** para poder hacer uso de la aplicación **Cardboard**. Además, es necesario que el *smartphone* disponga de giroscopio.

4.2 A nivel de Software

4.2.1 Unity

Una gran parte de los amantes de los videojuegos hemos soñado alguna vez con crear los suyos propios. Esto hasta hace relativamente poco era una tarea ardua y complicada. Sin embargo, en los últimos años han surgido herramientas que nos facilitan mucho esta labor.

Una de éstas es **Unity,** que junto con **UnrealEngine** es uno de los motores de desarrollo de videojuegos más importantes y usados del mundo. [11]

Se caracteriza por permitir realizar juegos multiplataforma (PC, Android, IOS, PS4, XBOX ONE, entre otros) tanto en dos como en tres dimensiones. El lenguaje utilizado para el scripting del mismo es C#, aunque no todo es programar, ya que disponemos de un editor gráfico que nos permite modelar los gráficos de éstos a nuestro gusto, así como de una tienda donde poder adquirir modelos ya creados si lo nuestro no es el diseño.

¿Por qué elegí Unity y no otro motor como el previamente mencionado Unreal Engine?

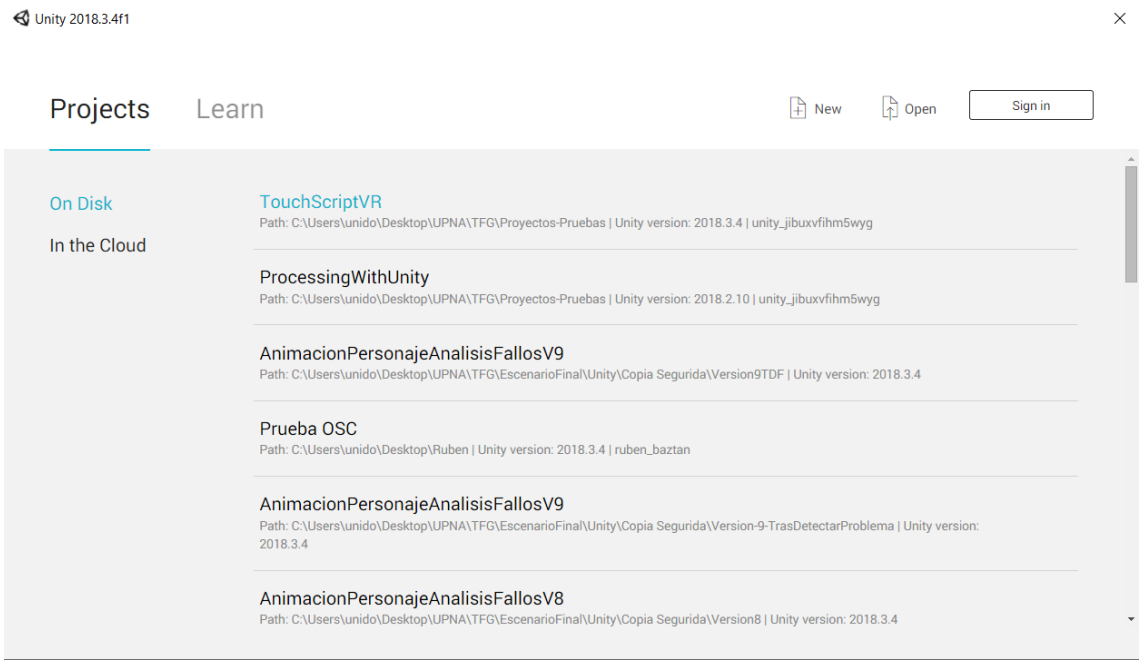
Cuando di la clase de **Sistemas Multimedia y Diseño Centrado en el Usuario** todos los proyectos que realizamos fueron creados con Unity, por lo que ya tenía una cierta base de conocimientos de éste. Con este trabajo se me abría una oportunidad de aumentar lo que aprendí en dicho curso sobre Unity, incluyendo los conceptos asociados a la realidad virtual.

A pesar de que otros motores de desarrollo también me ofreciesen todos aquellos recursos que necesitaba para cumplir que lo que se había ideado inicialmente para el juego, quería reforzar e incrementar lo que ya sabía previamente.

Explicación de la interfaz de Unity

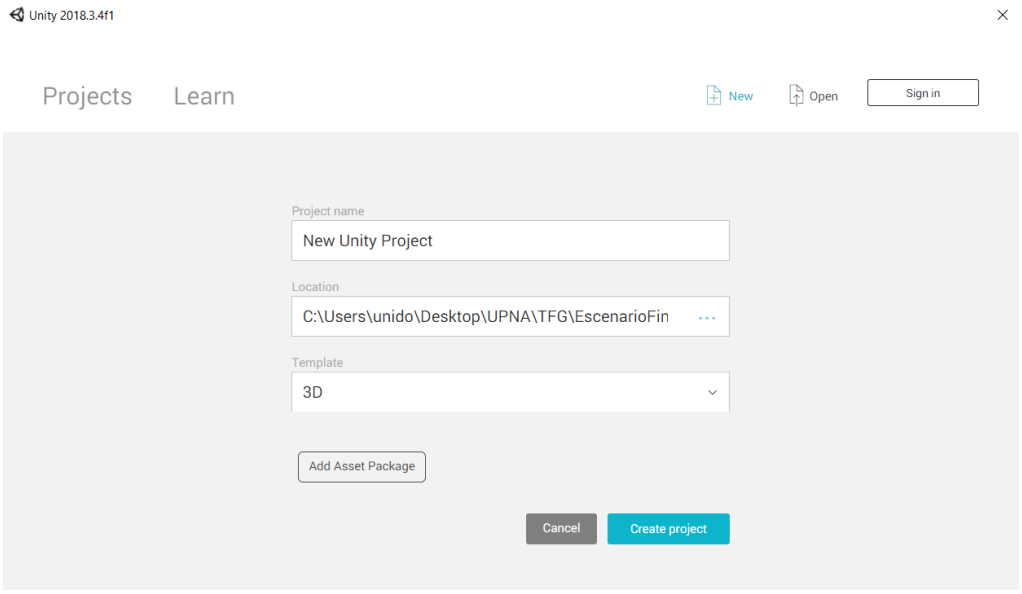
Debido al gran peso de este programa para la consecución del planteamiento inicial que teníamos del juego, he creído oportuno en explicar cada una de sus partes más en profundidad.

Nada más abrir el programa aparecerá una pantalla como la siguiente:



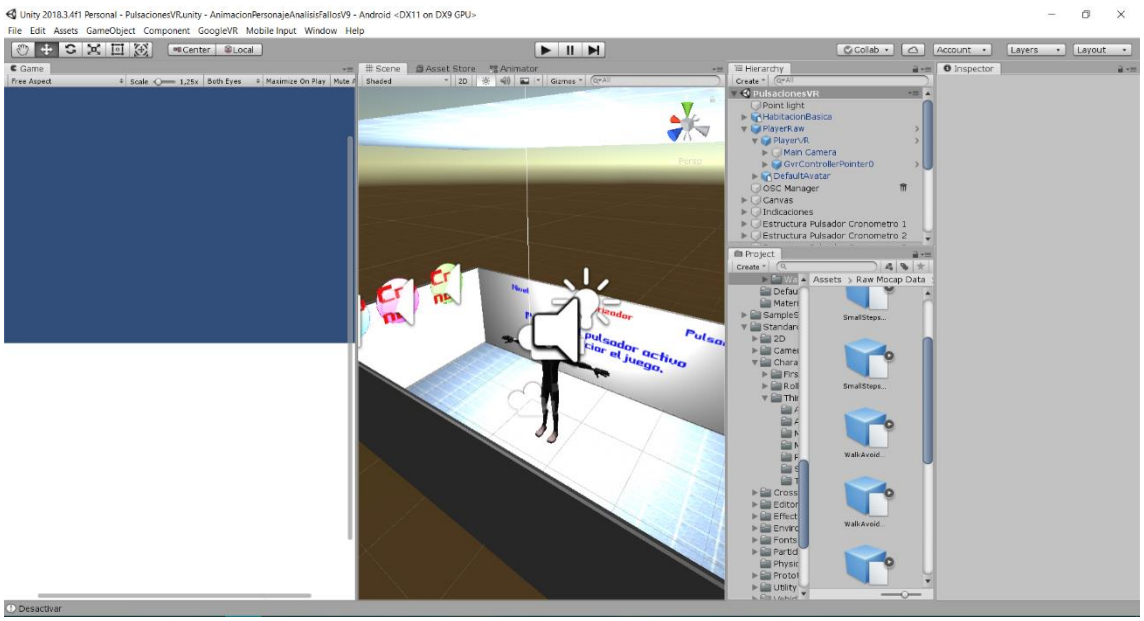
En ella se nos muestran dos pestañas llamadas **Projects** y **Learn.** La primera (en la que nos centraremos) nos permite crear o cargar tanto desde el ordenador como desde la nube proyectos. Por su parte, la pestaña restante nos presenta algunos tutoriales que nos enseñan los principios básicos que debemos conocer para poder desenvolvernos correctamente con el programa.

Si quisiéramos inicializar el desarrollo de un juego simplemente debemos de presionar el botón **New** y en la ventana emergente simplemente le debemos indicar al programa el nombre, la ruta y la dimensión que tendrá.



Para la explicación de cada una de las secciones del programa ya dentro del ámbito de desarrollo no partiremos desde un proyecto vacío, sino que las imágenes que aparecerán corresponder con Pulsaciones VR.

Dependiendo del *layout* que tengas establecido (si es la primera vez aparecerá el que está establecido por defecto) aparecerá una interfaz similar a la siguiente:

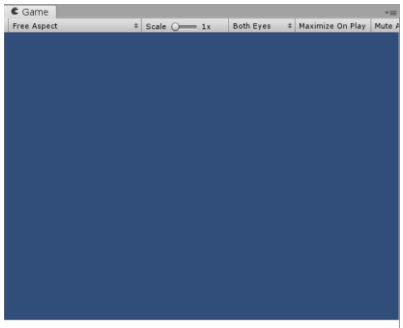


Como se puede contemplar podemos distinguir las siguientes áreas:

Ventana de Juego (Game)

En esta ventana podemos contemplar que es lo que verá la persona que disponga el juego y lo tenga en ejecución.

Puede verse como una previsualización de éste a medida que el desarrollo del mismo vaya en aumento.



Ofrece un menú desde el cual podemos entre otras cosas:

a. Aumentar la escala de visualización de la cámara.

b. Establecer la resolución de la cámara

c. En el caso de que la opción de Cardboard o Daydream este activado, permite visualizar el juego con la vista separada para cada uno de los ojos del jugador, solo con uno de ellos o sin hacer ningún tipo de distinción.

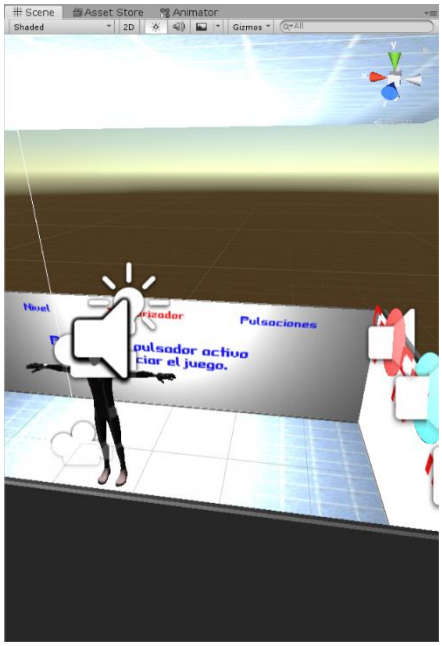
d. Contemplar el juego de manera que éste abarque una gran parte la interfaz de desarrollo activando la opción ***Maximize on Play.***

Ventana de Escena (Scene)

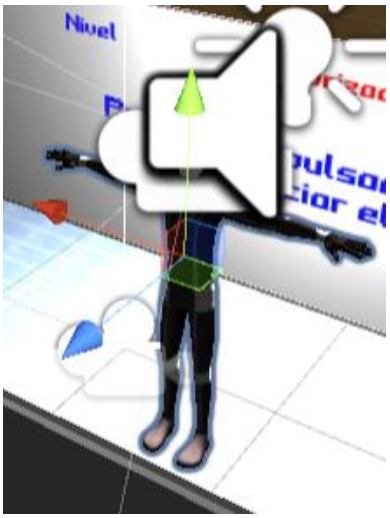
La vista de este apartado nos permite realizar una navegación y edición visual de los elementos del juego.

Puede mostrar una perspectiva del mismo tanto en dos como en tres dimensiones independientemente del tipo de juego que hayamos elegido en el momento en el que hemos creado el proyecto.

Como puede verse en la imagen asociada a este apartado algunos elementos distintivos, como pueden ser las cámaras utilizadas, los audios asignados a cada uno de los componentes que conforman la escena o la luz aplicada quedan remarcados sin necesidad de seleccionar el objeto que lo contiene en cuestión.

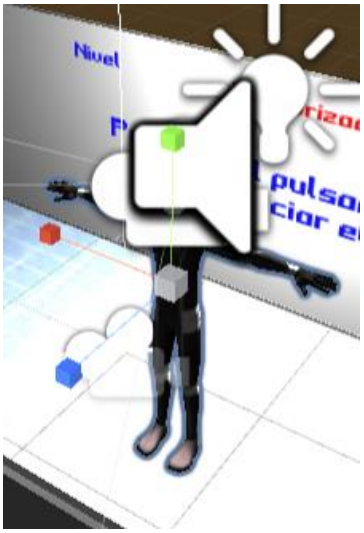


En función de que opción de las siguientes tengamos seleccionada en el editor, podremos desplazar, girar y modificar un objeto del juego que previamente hayamos presionado.

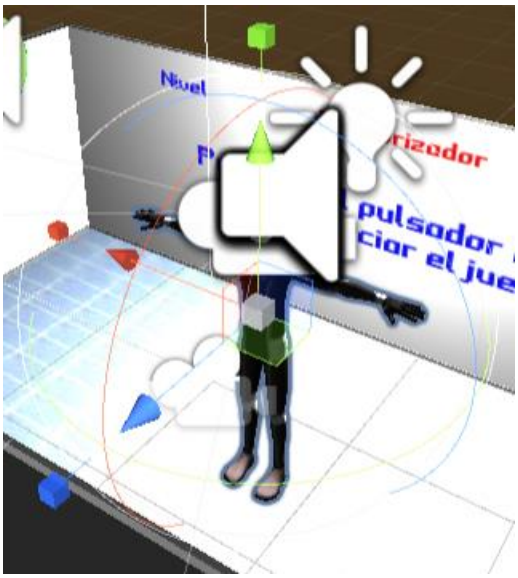


Desplazamiento

Giro



Deformación



Todas las opciones anteriores al mismo tiempo

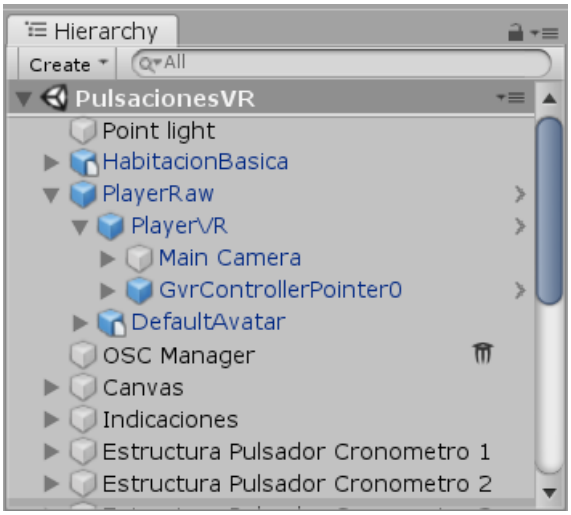
Ventana de Jerarquía (Hierarchy)

Sirve para mostrarnos todos los objetos que se encuentran dentro de una escena, es decir, en cada una de las secuencias del juego. Por ejemplo, el título de un juego puede estar en una escena mientras que cada uno de los niveles del mismo pueden tener la suya propia.

Esto en el caso de que los escenarios no cambien en exceso no es recomendable debido a la carga a nivel de memoria que supone ya que estaríamos clonando el mismo escenario en tantas escenas como niveles estén diseñados.

Es por eso por lo que, en el caso de Pulsaciones VR, aunque existan 4 posibles escenas, todo se construye en una sola. Para conseguir esto juego con activar o desactivar elementos de la escena en tiempo de ejecución.

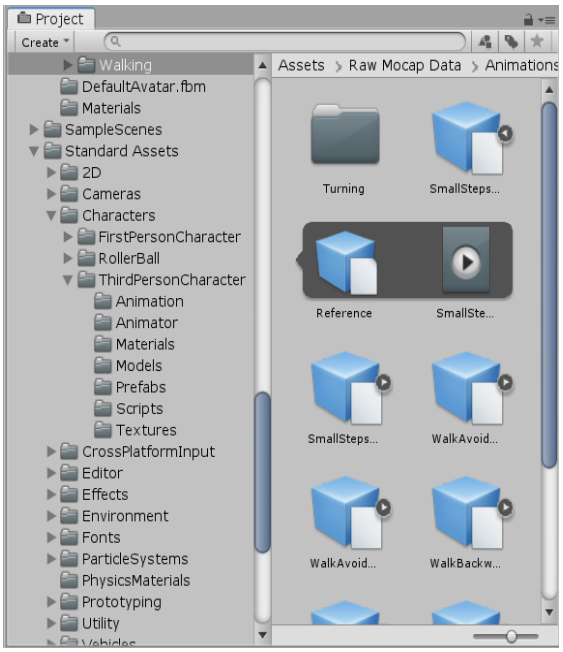
Para acabar con este punto cabe mencionar que cada uno de los elementos que conforman cada escena puede ser **padre** de otros y éstos a su vez de otros y así indefinidamente.



En el caso del proyecto que estamos explicando, el objeto **PlayerRaw** el cual representa al jugador tiene a su vez dos objetos hijos, los cuales representan a la cámara y al avatar que representa al jugador y de estos también parten más ramificaciones de elementos.

Observación: Más adelante explicaremos con detalle cada uno de los elementos que forman parte del proyecto.

Ventana de Proyecto (Project)

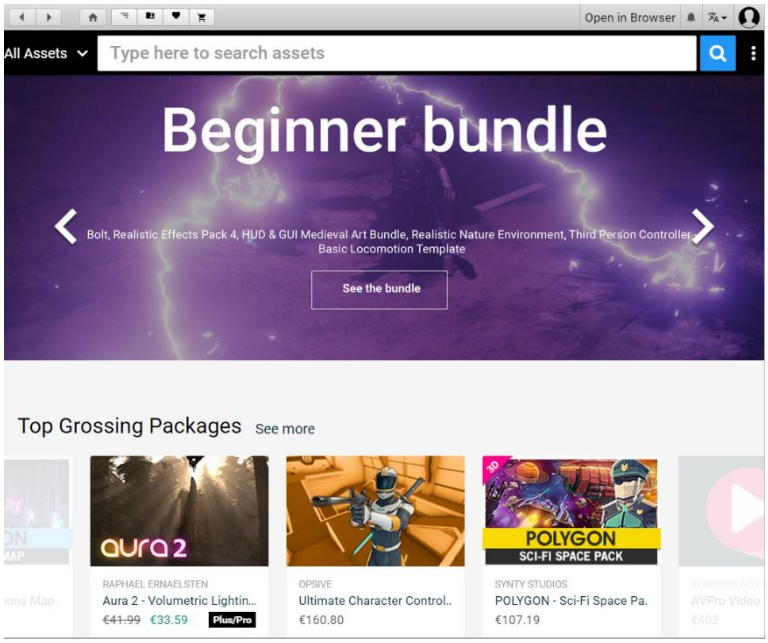


Ventana en las que se nos presentan dos áreas de las cuales la primera nos muestra un árbol con todas las carpetas que se encuentran dentro del proyecto, mientras que la segunda nos muestra el contenido de las mismas.

En la imagen que se adjunta pueden apreciarse en una de las carpetas alguno de los objetos guardados que forman parte o no de la escena los cuales se les conoce con el nombre de ***Asserts*** (o asertos)**.** Lo mencionado en el punto anterior sobre que existen elementos de la escena que son padres de otros también queda reflejado en este punto.

Asset Store

Para finalizar con el apartado de la explicación de las ventanas de Unity queda por presentar la tienda que pone a nuestra distribución Unity, en la cual podemos encontrar tanto de manera gratuita o de pago cualquier tipo de componente que necesitemos para nuestro proyecto.



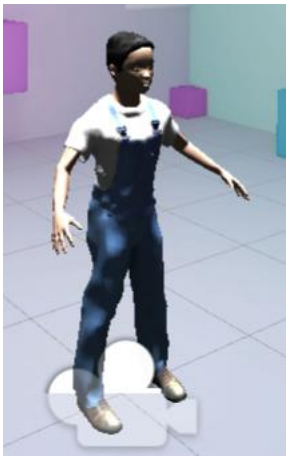
He decidido ponerlo como consecuencia de haber utilizado un par de paquetes para el desarrollo del proyecto:

a) Raw Mocap Data for Mecanim

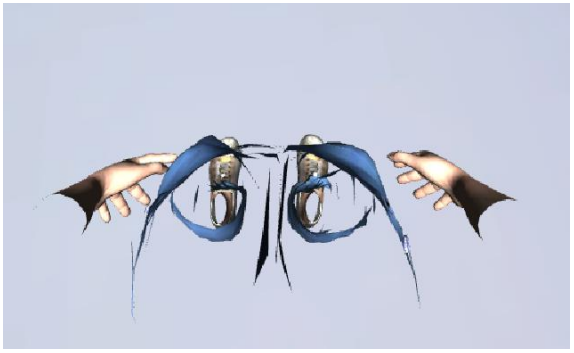
Desarrollado por el propio equipo de Unity, se trata de una colección la cual contiene elementos de animación de personajes.

En principio la idea que tenía era desarrollar un personaje en 3D desde 0, empleando el programa **MakeHuman** para el modelado y estructuración de los "huesos" del mismo para después emplear un paquete como el que estamos describiendo para darle vida por medio de animaciones. [12]

Sin embargo, a pesar de que las animaciones eran fluidas, la visión de las texturas desde la cámara principal con el modelo desarrollado por el programa no quedaba del todo bien.

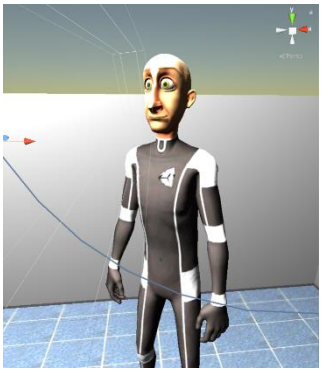


Modelo realizado con el programa MakeHuman

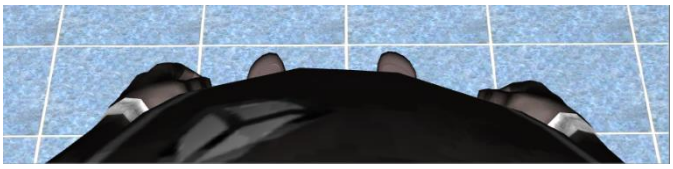


Vista del cuerpo del modelo desde la cámara situada en la cabeza del personaje

Es por eso por lo que me decanté por emplear el personaje por defecto que nos ofrece **Raw Mocap Data for Mecanim:**



Personaje Raw Mocap Data



Vista del cuerpo del modelo desde la cámara situada en la cabeza del personaje de Raw Mocap Data

b) extOSC – Open Sound Control

Desarrollado por Vladimir Sigalkin, se trata de una herramienta dedicada al desarrollo de aplicaciones que emplean el protocolo OSC (Open Sound Control). [13]

Para poder emplear este sistema, dentro de una misma red se nos permite establecer tanto **receptores** como **transmisores** de mensajes. Dentro de estos es necesario especificar tanto las IP como los puertos que queremos emplear.

Por otro lado, cada uno de los paquetes gestionados por este protocolo presenta una estructura compuesta por:

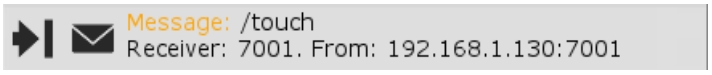
- **Dirección**
  Cadena semejante a la que nos podemos encontrar a la hora de mostrar la ruta hacia una carpeta. Sirve para distinguir diferentes tipos de mensajes dentro de un mismo proyecto.

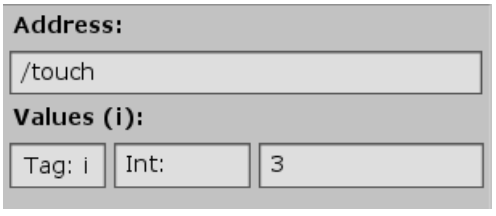
- **Tag**
  Puede definirse como el nombre de la variable que contiene alguno de los valores que se busca enviar en un mensaje.

- **Argumentos**
  La funcionalidad de este protocolo es la de enviar información. Es en este apartado en donde debemos de indicar que información es la que va a ser enviada o recibida.

Lo he empleado para la recepción en el juego de los mensajes OSC tanto de la posición del jugador como de la información referente a cuando los pulsadores son presionados.

Ejemplo de presentación del mensaje que se recibe al tocar el pulsador identificado con el valor 3 del juego:

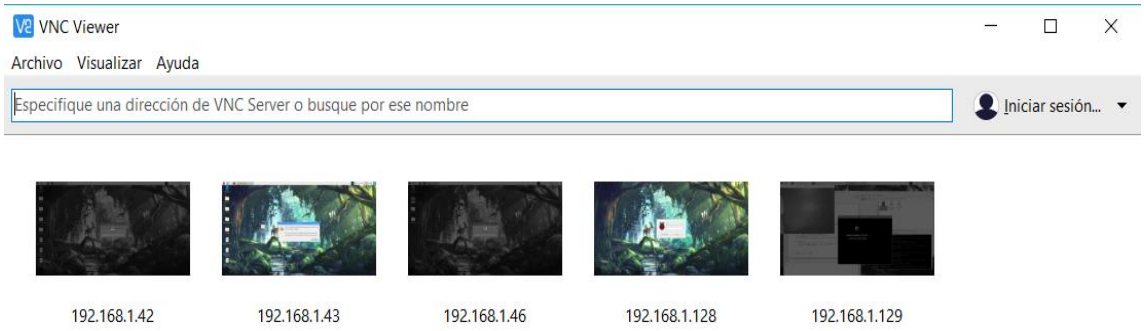




4.2.2 VNC Viewer

Como he comentado en el aparatado del hardware al hablar de Raspberry Pi, el método que había decidido emplear para conectarme a ella era mediante VNC (Virtual Network Computing), en concreto con el programa **VNC Viewer Remote Desktop.**

El proceso de conexión es sencillo, ya que únicamente es necesario indicar la IP del dispositivo del cual buscamos mostrar el escritorio de forma remota. Sin embargo, dicho dispositivo debe ser compatible con VNC, ya que en caso contrario el proceso no funcionará.



En el caso de que esto último mencionado no ocurriese, simplemente debemos de introducir el usuario y contraseña del dispositivo (si este tiene configurado algún tipo de seguridad a la hora de entrar al escritorio) y ya podremos interactuar con el sistema instalado en el mismo.

4.2.3 SweetHome3D

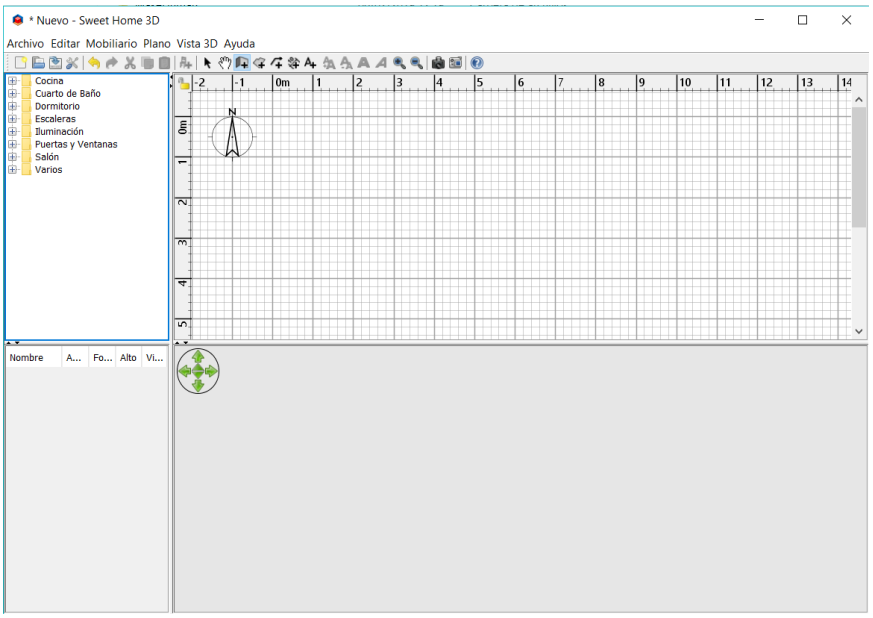
Con el fin de poder recrear la habitación en la que nos movemos de manera virtual disponía de dos opciones:

- Emplear las herramientas que me ofrece Unity para su obtención, ya sea importando salas creadas por terceros y descargadas desde la **Asset Store** o construyéndolas a partir de los elementos básicos de la interfaz de desarrollo.

- Utilizar aplicaciones especializadas en diseño de interiores que más tarde nos ofrezcan la posibilidad de exportar el proyecto que hayamos desarrollado a Unity.

Decidí decantarme por última opción, ya que en el caso de obtener la sala desde la tienda de Unity no me garantizaba que tuviera las dimensiones reales, y en el caso de hacerlo manualmente me resultaría más costoso de hacer. [14]

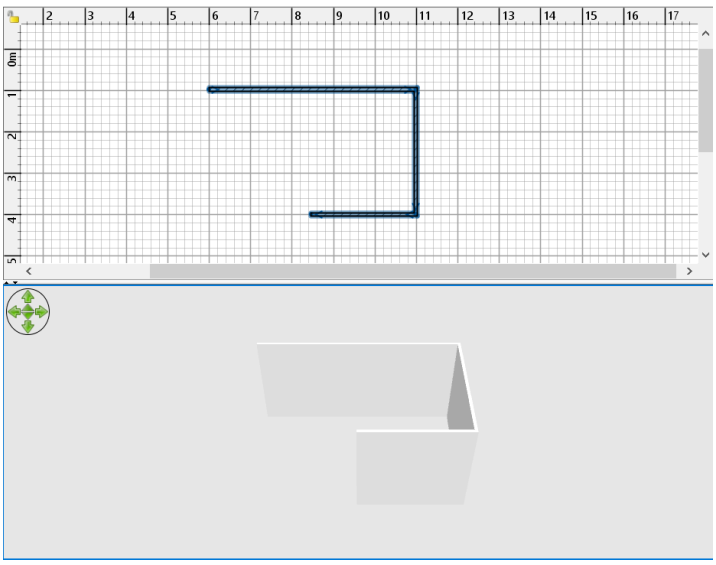
La aplicación más conocida que me permite cumplir con este objetivo es **SweetHome3D.** Su apariencia es la siguiente:



Para empezar a construir las paredes simplemente debemos de presionar el botón: 

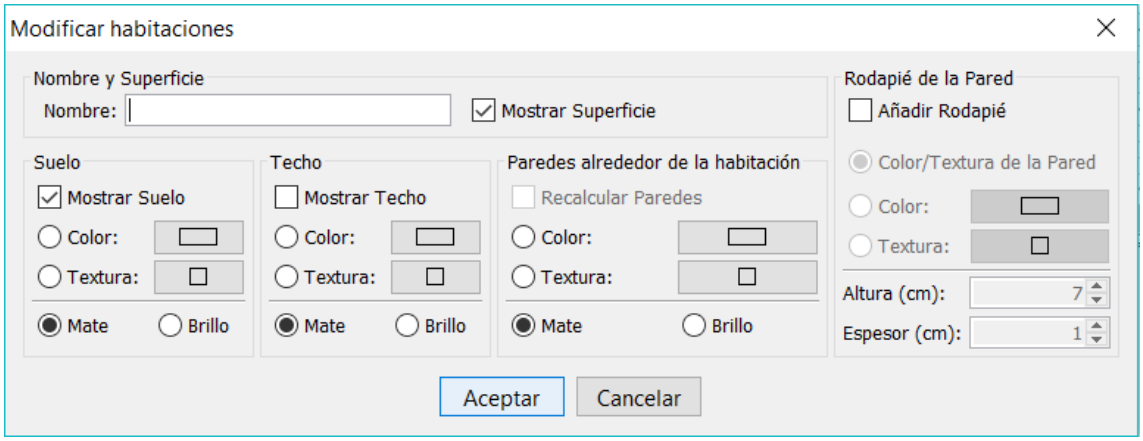
Una vez pulsado, debemos de presionar en la cuadrícula para ir determinando las esquinas de la sala que estamos modelando. Desde que clicamos por primera vez en ésta, nos aparecerá información sobre la distancia entre ese punto y la posición del ratón (distancia, grosor de la pared que está generando y el ángulo que forma con respecto al eje horizontal). Esto es importante ya que podemos representar la sala en su tamaño real.

A medida que vamos creando las distintas paredes de la habitación, se nos proporciona una vista previa en tres dimensiones de esta:



Una vez hemos puesto las paredes debemos crear la aplicación. Para ello utilizamos el botón que se encuentra a la izquierda del mencionado previamente ( ) pulsando en las cuatro esquina de la misma para realizar el cálculo del área.

Posteriormente podemos customizar la sala añadiéndole un techo o añadiéndole texturas al modelo. Para poder hacerlo debemos seleccionar el modelo, pulsar con el botón derecho del ratón sobre la cuadrícula y seleccionar la opción "**Modificar habitaciones**":



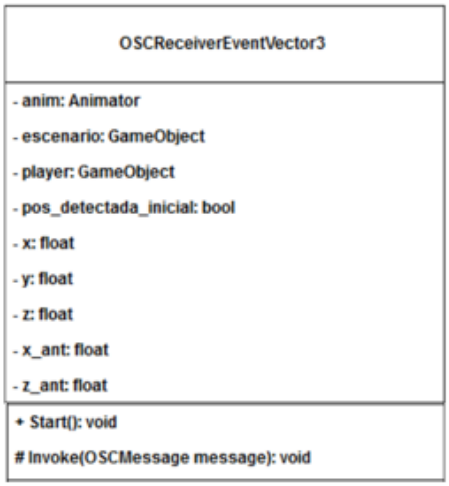
Finalmente, una vez tengamos el modelo terminado solo faltaría exportar el proyecto a Unity. Para hacerlo debemos de ir a la barra de herramientas y en la pestaña de "Vista 3D" escoger la opción "Exportar a Formato OBJ…" guardando el diseño dentro de la estructura de carpetas de nuestro proyecto desarrollado en Unity.

Una vez hecho esto podemos arrastrar la sala a la escena con la que estemos trabajando sin ningún problema y pudiendo acceder a cada uno de los elementos que la componen por si queremos realizar alguna modificación a nivel visual o para añadirles componentes como algún tipo de ***collider*** o ***rigidbody.***

5. DISEÑO DE LA APLICACIÓN

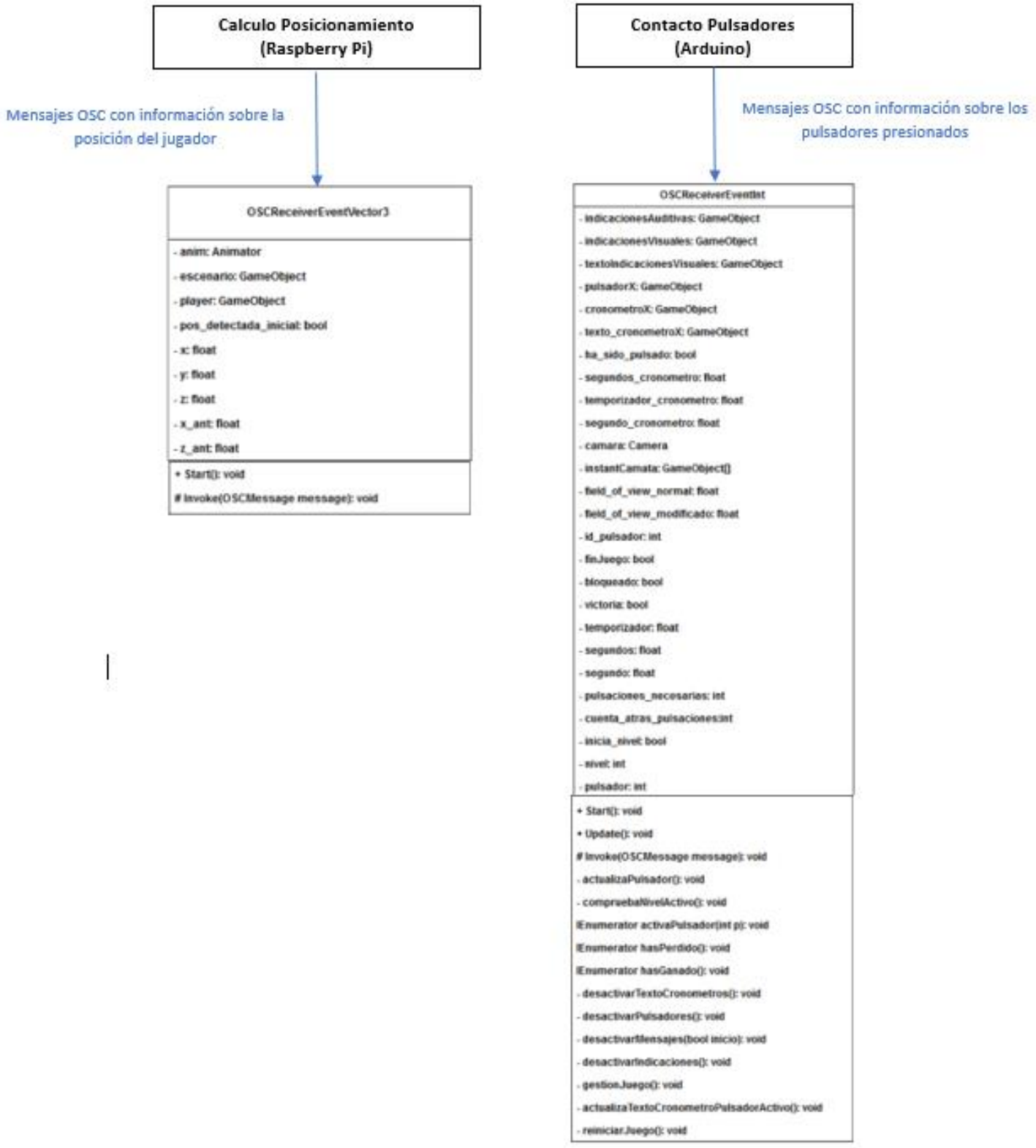
El objetivo del proyecto consistía en realizar un juego en realidad virtual que emplease la tecnología de cálculo de posicionamiento del jugador (mediante una Raspberry Pi), detección de contactos sobre pulsadores (mediante Arduino) y gestión del funcionamiento del juego mediante la recepción de mensajes de tipo OSC con la información necesaria para procesar los cambios producidos.

Los dos primeros sistemas tienen un código asociado, pero no se tratan de clases al uso. En cambio, en el código desarrollado en Unity (en lenguaje C#) si que podemos distinguir dos clases, las cuales son las encargadas de recibir y gestionar la información que es recibida en cada uno de los mensajes. Como únicamente he empleado clases procedentes del paquete OSCManager solo describiré aquellas que he modificado para que el programa funcione de acuerdo a la idea de juego planeada. Las clases serían las siguientes:

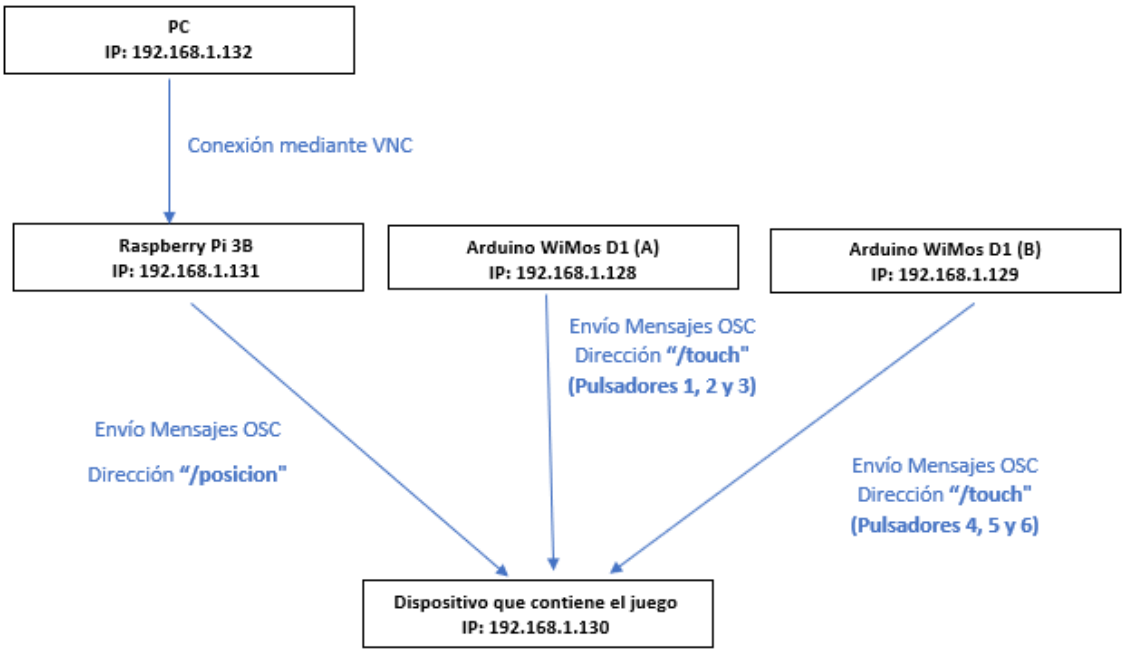


Observación: Con respecto a la clase OSCReceiverEventInt decir que algunas de las variables que eran muy parecidas han sido agrupadas para que la tabla pudiese entrar en una única hoja del documento (por ejemplo, se ha definido la variable pulsadorX en vez de poner pulsador1, pulsador2…).

Teniendo en cuenta estas dos clases, la comunicación entre los tres sistemas sería de la siguiente manera:



Los diferentes sistemas se encuentran en la misma red para poder llevar a cabo la comunicación. El esquema de IP empleado es el siguiente:



6. IMPLEMENTACIÓN DEL PROYECTO

En este apartado del documento nos vamos a centrar en cómo se ha llevado a cabo del desarrollo del juego explicando en su totalidad los tres puntos más importantes del mismo, los cuales son el **cálculo del posicionamiento del jugador, gestión de los contactos sobre los pulsadores y la implementación del juego en Unity.**

6.1 Cálculo de la posición del usuario

La detección del movimiento del jugador la considero como el punto más complicado de realizar. Esto lo como consecuencia de no haber elegido la rama de computación en la cual se dan conceptos sobre el tratamiento de imágenes.

Uno de los lenguajes más empleados para este tipo de proyectos gracias a la gran cantidad de librerías dedicadas a esta temática es Python. Como consecuencia de lo mencionado en el párrafo anterior no había tenido la oportunidad de conocerlo en profundidad.

Sin embargo, gracias a la gran comunidad de personas relacionadas con este lenguaje y a la documentación que ofrecen sobre el mismo y sus librerías han hecho que este frente no fuera tan tedioso que cuando me lo imaginaba a la hora de idear el juego.

6.1.1 Librería OpenCV

Antes de mostrar y explicar el código asociado a este punto, creo que es conveniente presentar OpenCV (**Open Source Computer Vision**) la cual es la encargada de ofrecernos las herramientas necesarias para conseguir tratar las imágenes correctamente en función a nuestras necesidades. [15]

Se trata de una librería de código abierto escrita en C/C++ y desarrollada por Intel, cuya primera versión fue lanzada a mediados del año 2000, aunque actualmente van por la 4.1.0. Es multiplataforma y puede utilizarse con multitud de lenguajes de programación entre los que se encuentra multitud de ejemplos tales como C#, Java y el previamente mencionado Python.

A pesar de desconocer el lenguaje me decanté por Python no solo por obligarme a conocer un nuevo lenguaje, sino también por su gran **portabilidad** algo a tener muy en cuenta debido a que teníamos que ejecutar el programa desde una Raspberry Pi.

Volviendo a las características de OpenCV decir que su principal objetivo es el de facilitar el trabajo de aquellos programadores dedicados al desarrollo de programas relacionados con el mundo de la visión artificial poniendo a su disposición multitud de funciones orientadas entre otras cosas al reconocimiento facial o lo que nos atañe para nuestro proyecto, la **detección de movimiento**. [16][17]

En el proceso de instalación de Python y OpenCV que aparece en el enlace asociado al punto donde son explicadas las características de la Raspberry Pi 3 se indica que es necesario escribir en una terminal el comando **workon cv** con el fin de poder acceder al entorno virtual de Open CV.

6.1.2 Explicación del código

Antes de empezar a explicar cada una de las partes del código asociado a la detección de la posición del jugador creo que es conveniente tener una visión global del mismo:

```
import numpy as np
import cv2
import OSC

client = OSC.OSCClient()
client.connect(("192.168.1.130", 7002))

camara = cv2.VideoCapture(0)
fondo = None

i=0

while i<25:
    (grabbed, frame) = camara.read()
    i=i+1


while True:
    (grabbed, frame) = camara.read()

    if not grabbed:
        break

    gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gris = cv2.GaussianBlur(gris, (21, 21), 0)

    if fondo is None:
        fondo = gris
        continue

    resta = cv2.absdiff(gris, fondo)
    umbral = cv2.threshold(resta, 25, 255, cv2.THRESH_BINARY)[1]
    umbral = cv2.dilate(umbral, None, iterations=2)
    contornoimg = umbral.copy()
    im, contorno, hierarchy = cv2.findContours(contornoimg, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    for c in contorno:
        if cv2.contourArea(c)<6000:
            continue

        (x,y,w,h) = cv2.boundingRect(c)
        cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0), 3)
        cv2.circle(frame, ((x+x+w)/2, (y+y+h)/2), 5, (0,255,0), -1)

        msg = OSC.OSCMessage()
        msg.setAddress("/posicion")
        msg.append((x+x+w)/2)
        msg.append((y+y+h)/2)
        client.send(msg)

        cv2.imshow("Camara", frame)
        key = cv2.waitKey(1) & 0xFF

        if key == ord("s"):
            break

    camara.release()
    cv2.destroyAllWindows()
```

Librerías

Las librerías que hemos utilizado son las siguientes:

- **import numpy as np**

  Librería que se encarga de dar un gran soporte al entorno de Python a la hora de realizar el cálculo de vectores y matrices. Sin ella no sería posible calcular y almacenar la información asociada a los diferentes contornos que el programa detecta.

- **import cv2**

  Librería que contiene todas las funciones que necesitamos para tratar la imagen para poder obtener las coordenadas del punto central del contorno del jugador.

- **import OSC**

  Librería necesaria para poder crear y enviar los mensajes de tipo OSC al dispositivo móvil que está ejecutando el juego.

Establecimiento de conexión con el dispositivo móvil

Para poder llevar a cabo la comunicación por medio de mensajes de tipo OSC es necesario indicarle al programa la IP que identifica al *smartphone* y el puerto que está empleando para la recepción de estos.

```
client = OSC.OSCClient()
client.connect(("192.168.1.130", 7002))
```

Obtención de las imágenes a tratar

Las imágenes son obtenidas de la cámara conectada a la Raspberry Pi. Con el fin de poder tomar imágenes en tiempo real simplemente debemos de iniciar la grabación de la cámara.

```
camara = cv2.VideoCapture(0)
```

Inicialización de la variable que contendrá al primer fotograma

La idea para localizar al usuario consiste básicamente en comparar una imagen en la que se muestra la sala vacía con las sucesivas capturas que la cámara va realizando.

Es necesario almacenar ese primer *frame* en una variable (que hemos llamado **fondo**) para posteriormente trabajar con ella. Sin embargo, en este punto del código es inicializado a "None" porque la cámara necesita un tiempo de refresco para sacar imágenes con un contraste ya definido.

```
fondo = None
```

Primeras capturas

Como consecuencia de que al inicializar el programa encendemos a la par la cámara, las primeras capturas presentan diferentes tipos de contrastes hasta que logra adaptarse a la luz de la habitación. Es por eso por lo que se debe realizar un primer bucle en el que la cámara se estabilice.

```
i=0

while i<25:
    (grabbed, frame) = camara.read()
    i=i+1
```

Observación: Las siguientes partes del código en ser comentadas forman parte del bucle que se ejecuta hasta que se fuerza su salida presionando la tecla 's' o realizando la detección del proceso por medio de la combinación de teclas "Control + C" o "Control + Z".

Almacenaje de las capturas realizadas por la cámara, escala de grises y suavizado

Como se ha mencionado en puntos anteriores, la idea del programa es la de comparar una imagen inicial con respecto a las siguientes que van siendo tomadas para analizar posibles diferencias entre unas y otras dándonos la idea de que se ha producido algún tipo de movimiento.

```
(grabbed, frame) = camara.read()

if not grabbed:
    break

gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gris = cv2.GaussianBlur(gris, (21, 21), 0)

if fondo is None:
    fondo = gris
    continue
```

Mediante la función ***read()*** vinculado a la cámara podemos analizar si la cámara a logrado tomar alguna instantánea y de ser así, inicarnos cual es. Estos dos valores los he almacenado en las variables "***grabbed"*** y ***"frame"*** respectivamente.

En caso de que el valor almacenado en la primera de estas sea "false" nos indica que se ha producido un error en el proceso de captura de imágenes. En el caso de que esto se manifieste debemos de salir del bucle para no se produzca ningún tipo de error al analizar una variable la cual no tiene ningún tipo de valor representativo como es el caso de ***"frame".***

Sin embargo, en el caso de que la captura sea realizada correctamente, el siguiente paso a realizar sería el de pasar esta a **escala de grises**. Esto lo hacemos como consecuencia de para nuestro propósito de realizar la diferencia de dos capturas es óptimo y menos complejo de llevar a cabo.

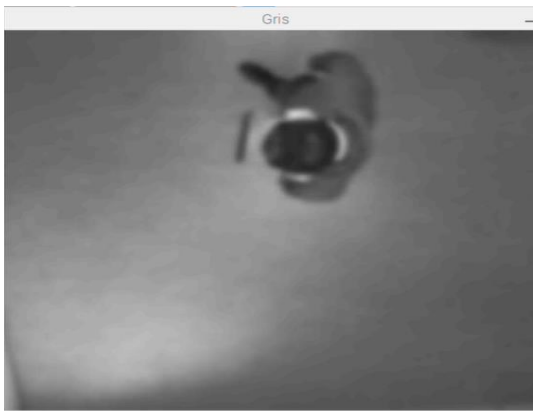
Si no lo hiciéramos cualquier tipo de diferenciación en la iluminación sería más apreciable, haciendo que el proceso de sustracción de todo aquello que se repite en las capturas comparadas fuese menos precisa.

Esto también ayuda en el siguiente paso necesario el cual consiste en "suavizar" las instantáneas que van siendo tomadas, es decir, promediar todos aquellos píxeles vecinos muy parecidos entre sí. El fin de esto consiste en minimizar la mayor cantidad de ruido que es originado.

Finalmente decir que, si aún no se dispone de una primera captura con la que ir realizando la comparación es preciso guardarla.

Los resultados obtenidos al ir realizando la ejecución de ese segmento de código serían los siguientes:

Primer frame con la habitación vacía



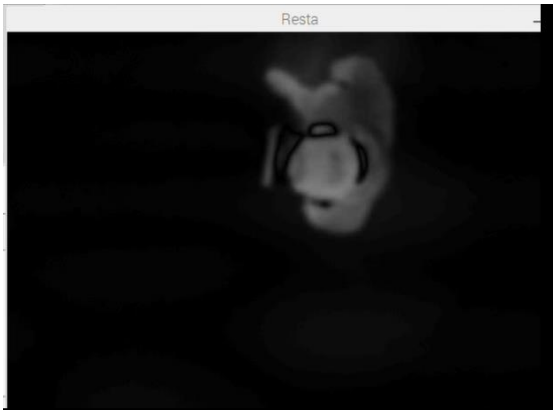
Uno de los sucesivos frames ya con el jugador moviéndose en el entorno.

Detección de cambios entre imágenes

Una vez que disponemos de dos imágenes las cuales comparar ya podemos ejecutar el proceso de diferenciación entre las mismas. Para ello nos apoyamos en las siguientes funciones:

```
resta = cv2.absdiff(gris, fondo)
umbral = cv2.threshold(resta, 25, 255, cv2.THRESH_BINARY)[1]
umbral = cv2.dilate(umbral, None, iterations=2)
```

Lo primero que debemos realizar es la resta entre las dos imágenes presentadas en el punto anterior mediante el método "***absdiff"*** obteniendo el siguiente resultado:



Como puede apreciarse, todos aquellos píxeles que coinciden en ambas capturas, al ser suprimidos hacen que tomen el color negro. En caso contrario, al no coincidir éstos se nos muestra el cambio entre las mismas en un rango de tonalidades en escala de grises.

Dependiendo del parecido de color entre las prendas de vestir y el suelo a la hora de realizar el cambio de color puede hacer que el efecto que obtengamos no sea muy significativo, lo cual puede resultar un problema si no lo tenemos en cuenta.

Para que a la hora de calcular el contorno no tengamos ningún tipo de problemas vinculado a las distintas tonalidades de la persona detectada, el siguiente paso es destacar todos aquellos píxeles que no sean completamente negros y que superen un cierto umbral, dándoles a estos un determinado color, en nuestro caso blanco. Para llevar a cabo esto empleamos las funciones "***threshold"*** y ***"dilate"***, dando lugar a:

Cálculo del contorno y envío al juego del punto central de la diferencia

Una vez disponemos de la diferencia entre las dos imágenes, el siguiente paso es el de obtener el contorno de este para después calcular el punto central del mismo. Este paso puede realizarse de múltiples formas, desde ceñirnos propiamente a los contornos y trabajar con todos ellos, o contener estos en un área y trabajar con la misma.

Decidí decantarme por la segunda opción introduciendo todos los contornos en tipo de figura geométrica con la cual trabajar (en concreto los agrupé en un rectángulo), del cual era sencillo calcular el punto central.

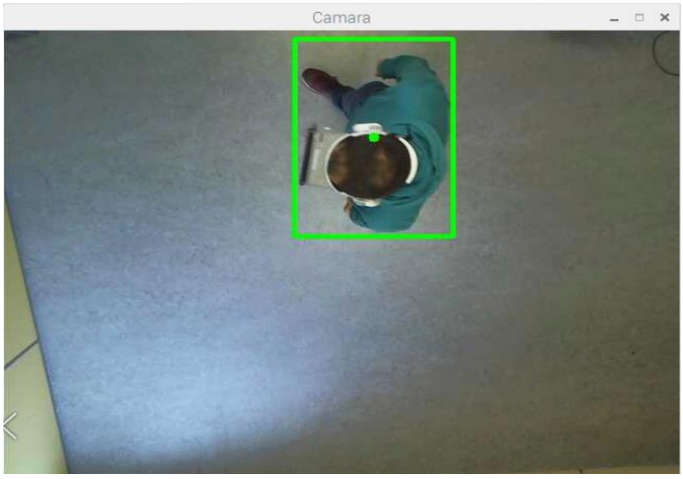
El código para este proceso es el siguiente:

```
contornoimg = umbral.copy()
im, contorno, hierarchy = cv2.findContours(contornoimg, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

for c in contorno:
    if cv2.contourArea(c)<6000:
        continue

    (x,y,w,h) = cv2.boundingRect(c)
    cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0), 3)
    cv2.circle(frame, ((x+x+w)/2, (y+y+h)/2), 5, (0,255,0), -1)

    msg = OSC.OSCMessage()
    msg.setAddress("/posicion")
    msg.append((x+x+w)/2)
    msg.append((y+y+h)/2)
    client.send(msg)

    cv2.imshow("Camara", frame)
    key = cv2.waitKey(1) & 0xFF

    if key == ord("s"):
        break
```

En primer lugar, hacemos una copia del umbral previamente obtenido. Este paso es opcional ya que únicamente lo he realizado para más adelante poder mostrar todos los tipos de representaciones los "*frames*" que están siendo procesados a la vez.

Posteriormente empleamos la función "***findContours***" para obtener cada uno de los contornos detectados. Una vez los tengamos despreciamos todos aquellos que tengan un tamaño menor a un valor que consideremos pequeño y que pueda llegar a interferir en el correcto funcionamiento de la detección del jugador. De esta manera tratamos de delimitar a quedarnos con el contorno más grande el cual represente al cuerpo de este.

Una vez lo localicemos, con la función "***boundingRect***" mediante la cual se calculan las coordenadas que más tarde nos permitirán representar visualmente tanto el rectángulo que engloba el contorno, como el punto central del mismo.



Ya calculado y representado el punto central del contorno ya solo queda enviarlo al juego mediante mensajes de tipo OSC. Previamente ya habíamos indicado tanto la IP como el puerto de destino, por lo que solo quedaría crear un mensaje indicando una dirección (en nuestro caso le damos el valor **posicion**), y a dicho mensaje le adherimos las coordenadas del punto.

Liberar cámara y cerrar ventanas de visionado de la situación

Al salir del bucle únicamente nos quedaría terminar con los procesos asociados al programa, en concreto nos quedaría liberar la cámara y cerrar todas las ventanas que nos han servido para representar cada una de las funciones que hemos empleado para modificar las imágenes hasta poder llegar a cumplir el objetivo de obtener la posición del jugador dentro de la sala.

```
camara.release()
cv2.destroyAllWindows()
```

6.1.3 Desventajas de la idea planteada

Aunque el planteamiento del sistema previamente explicado funciona correctamente, sí que es cierto que no es perfecto y que tiene algunos aspectos a mejorar que por diferentes razones no las he podido llevar a cabo.

- **Es necesaria una habitación con una gran altura y una cámara que pueda captar en su totalidad la misma.**

  Para poder simular correctamente el movimiento del jugador dentro del juego es necesario poder capturar toda la sala. Para el desarrollo del proyecto la cámara que empleé fue situada a una altura comprendida entre los 4 y los 5 metros. Aun así, el suelo de la sala no es detectado en su totalidad.

Imagen de la cámara situada cerca del techo de la habitación

- **Únicamente puede ponerse en la sala un único jugador**

  A pesar de poder detectar múltiples contornos, si ponemos varios jugadores dentro de una misma sala, la recepción del punto central sería caótico como consecuencia de obtener varios puntos centrales a la vez.

  Esto haría que, de manera aleatoria ya que depende de que mensaje sería recibido en primer lugar, cada jugador, aunque esté en una posición distinta a otro puede situarse virtualmente donde este último.

- **Interacciones con otros elementos de la sala**

  Relacionado con el punto anterior, si en el transcurso del juego cualquier objeto es desplazado, el programa detectaría que se ha producido un desplazamiento, enviando dos informaciones al jugador, la información de su posición y la del objeto que es desplazado.

- **Un cambio en el contraste de la habitación puede hacer que la ejecución del juego no sea correcta.**

  A lo largo de la explicación de la detección del posicionamiento de la persona hemos mencionado que comparamos cada una de las imágenes que la cámara va realizando con una inicial.

  Sin embargo, si la iluminación de una habitación después de obtener ese primer fotograma cambia, haría que se detectase que el cambio entre una imagen y otra fuese total.

- **Es detectado el centro del contorno, no la verdadera posición del usuario**

  Como se puede apreciar en la imagen en la que aparece delimitado el contorno del jugador en un recuadro del color verde, el centro parece ubicarse más o menos donde se encuentra la cabeza. Sin embargo, si en algún momento las dimensiones del recuadro se verían modificados, por ejemplo, al estirar los brazos, el centro sería distinto.

  Esto hace que, si estuviéramos en el juego parados, al alargar los brazos hacia en frente tendríamos la sensación como de si hubiéramos dado un pequeño paso adelante, cuando esto no debería ser así.

- **La imagen captada por la cámara no capta las verdaderas dimensiones de la habitación**

  Cuando los diferentes fotogramas son tomados por la cámara presentan unas ciertas dimensiones (más o menos 550x550). A la hora de llevar esta información al juego no se adapta de manera correcta al entorno virtual teniendo que adaptar esta información para que la emulación del movimiento sea medianamente correcta.

6.1.4 Alternativa al desarrollo: Detección por color

Una alternativa al planteamiento anterior consistía en localizar al usuario por medio de la detección de un color específico. No dependeríamos de un primer fotograma para ir comparando evitándonos los problemas que este conlleva.

La idea consistiría en llevar una pegatina de un color muy distintivo situada en la cabeza. Esto nos permitiría entre otras tener más de un usuario a la vez al jugar con diferentes colores, el poder desplazar objetos sin temor a que la ejecución del juego se viese afectada y el movernos como queramos sin que el contorno se viese modificado y por lo tanto la información que es posteriormente enviada.

El código de este planteamiento alternativo es el siguiente:

```
import cv2
import numpy as np
import imutils

camara = cv2.VideoCapture(0)

while True:
    grabbed, frame = camara.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    color_intenso = np.array([80, 255, 255])
    color_apagado = np.array([50,50,50])

    mascara = cv2.inRange(hsv, color_apagado, color_intenso)
    mascara = cv2.erode(mascara, None, iterations=2)
    mascara = cv2.dilate(mascara, None, iterations=2)

    # Al mostrar la mascara vemos como
    #result = cv2.bitwise_and(frame, frame, mascara-mascara)
    cv2.imshow("Resultado", mascara)

    #diferencia = cv2.cvtColor(mascara, cv2.COLOR_BGR2GRAY)
    im, contorno, hierarchy = cv2.findContours(mascara, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    #contorno = imutils.grab_contours(contorno)
    centro = None

    if len(contorno)>0:
        c = max(contorno, key=cv2.contourArea)
        ((x,y), radio) = cv2.minEnclosingCircle(c)
        M = cv2.moments(c)
        centro = (int(M["m10"]/M["m00"]), int(M["m01"]/M["m00"]))

        if radio>10:
            cv2.circle(frame, (int(x), int(y)), int(radio), (0, 255, 255), 10)
            cv2.circle(frame, centro, 5, (0,0,255), -1)



    #cv2.imshow("Diferencia", diferencia)
    cv2.imshow("Frame", frame)



    if cv2.waitKey(1) == 27:
        break

camara.release()
cv2.destroyAllWindows()
```

Sin embargo, no voy a realizar una explicación de esta ya que se decidió descartar esta opción debido a dos fuertes problemas que tenía:

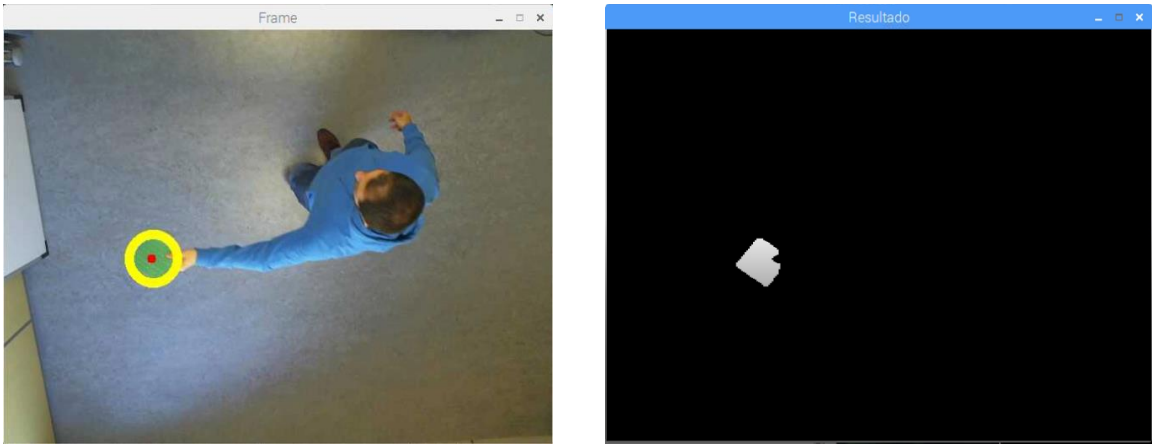
- **Gran dependencia del contraste de la habitación**

  Aunque se busca trabajar con una amplia gama de tonalidades dentro del color que se quería detectar, como consecuencia de las condiciones de iluminación de la sala había momentos en los que el elemento a encontrar se localizaba muy bien y otras veces ni aparecía.

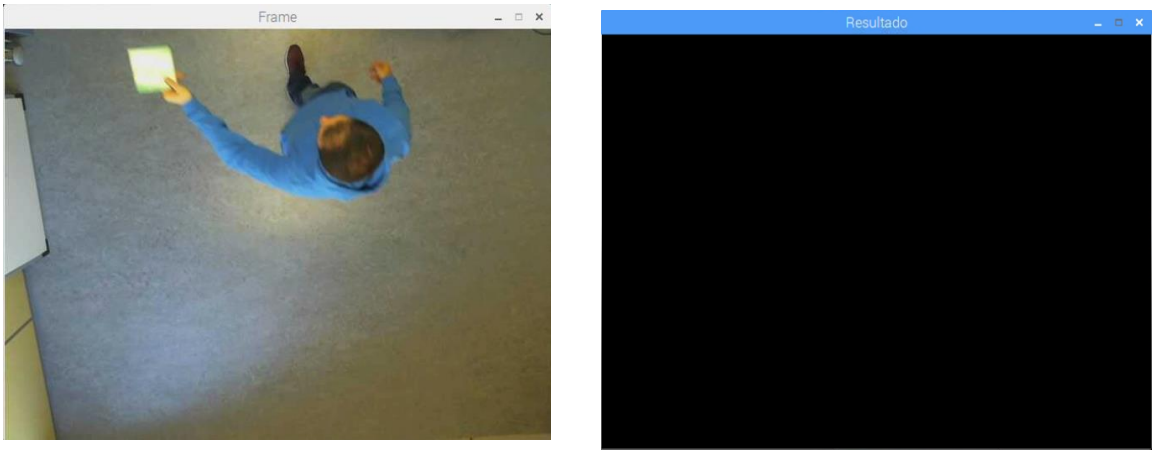
- **Problemas al mover la cabeza**

  Al estar localizado el elemento a detectar en la cabeza del jugador, en el momento en el que esta era inclinada más de la cuenta provocaba que la cámara tuviera problemas para indicar su posición.

A continuación, se adjuntan un par de situaciones asociadas a las pruebas de detección de color en el que se pueden contemplar los graves problemas antes mencionados. Al girar el objeto a detectar, la iluminación hace que el color de este varíe imposibilitando su detección.



Situación en el que el color del elemento a detectar es localizado correctamente



Situación en la que se manifiesta un problema con la iluminación y el objeto deja de ser detectado por la cámara

6.2 Gestión de los contactos sobre los pulsadores

Para poder entender cómo funciona la gestión de los pulsadores al ser presionados, y como se realiza el envío de mensajes, lo mejor es ver el código que hace que el sistema funciones.

```
/*************************************************************
 * Librerías
 *************************************************************/
#include <ArduinoOSC.h>
#include <Wire.h>
#include "Adafruit_MPR121.h"

/*************************************************************
 * Variables utilizadas
 *************************************************************/
WiFiUDP udp;
ArduinoOSCWiFi osc;
// Indicamos a que red vamos a conectarnos junto con la ip
// a la que mandaremos los paquetes de tipo OSC
const char* ssid = "labinteractivos";
const char* pwd = "labinteractivos";
const char* host = "192.168.1.130";

// Establecemos el puerto que emplearemos a la hora de mandar el mensaje
//const int recv_port = 7005;
const int send_port = 7002;

// Finalmente inicializamos los valores de las variables
// que representaran a los últimos pulsadores presionados
Adafruit_MPR121 cap = Adafruit_MPR121();
uint16_t lasttouched = 0;
uint16_t currtouched = 0;

/*************************************************************
 * Función setup()
 *************************************************************
 * Primera función en ejecutarse dentro de un programa Arduino
 * Establecemos comunicación serial y realizamos la configuración
 * vía WIFI con el destinatario de los mensajes
 *************************************************************/
void setup(){
    // En primer lugar tratamos de conectarnos por WIFI a la red
    delay(2500);
    Serial.begin(115200);
    WiFi.disconnect(true);
    WiFi.begin(ssid, pwd);

    // Hasta que no pueda llevar a cabo la conexión, el programa se quedará esperando
    while (WiFi.status() != WL_CONNECTED){
        delay(500);
    }

    // En el momento en el que se conecta, mostramos el siguiente mensaje indicándolo:
    Serial.print("WiFi connected! IP address: ");
    Serial.println(WiFi.localIP());

    // Establecemos el inicio de mensajes de tipo OSC
    osc.begin(udp, send_port);
```

```
    // La dirección predeterminada es 0x5A. (Configuración empleada)
    // En el caso de estar vinculada a 3.3V, deberá de ser 0x5B.
    // Si está vinculado a SDA deberá ser 0x5C
    // Si en cambio lo está a SCL deberá ser 0x5D
    if (!cap.begin(0x5A)) {
      while (1);
    }

    // Mandamos un primer mensaje para indicar que se inicializa
    // la emisión de mensajes.
    OSCMessage msg;

    msg.beginMessage(host, send_port);
    msg.setOSCAddress("/init/");
    osc.send(msg);
}

/**************************************************************************
 * Función loop()
 **************************************************************************
 * Función ejecutada por Arduino infinitas veces.
 * En este método buscamos realizar la gestión del envío de mensajes
 * cada vez que uno de los pulsadores conectados a la placa es
 * presionado.
 **************************************************************************/
void loop(){
    // Si logramos establecer la conexión empleamos el método loop() para
    // mandar mensajes a medida que se vayan detectando pulsaciones.
    OSCMessage msg;
    // Indicamos cual será el receptor de los mensajes.
    msg.beginMessage(host, send_port);

     // Realizamos la gestión de los 12 controladores de la placa capacitiva
    currtouched = cap.touched();
    int id=0;
    for (uint8_t i=0; i<12; i++) {

      // Si tocamos cualquiera d elos pulsadores se enviará el mensaje
      // al destinatario. Aparecerá en la consola del IDE de Arduino
      // un mensaje indicando que pulsador esta siendo presionado.
      if ((currtouched & _BV(i)) && !(lasttouched & _BV(i)) ) {
          Serial.print(i); Serial.println(" touched");
          // Indicamos la dirección que tendrá el mensaje
          char myAddress[80];
          char myCharArray[]="/touch";

           msg.setOSCAddress(myCharArray);
           msg.addArgInt32(id);

          //Realizamos el envío
          osc.send(msg);
          print_capacitive();
      }
      // En el momento que se deja de pulsar, aparecerá un mensaje
      // indicándolo
      if (!(currtouched & _BV(i)) && (lasttouched & _BV(i)) ) {
        Serial.print(i); Serial.println(" released");
      }
      id++;
    }

    // Reestablecemos el estado
    lasttouched = currtouched;

}
```

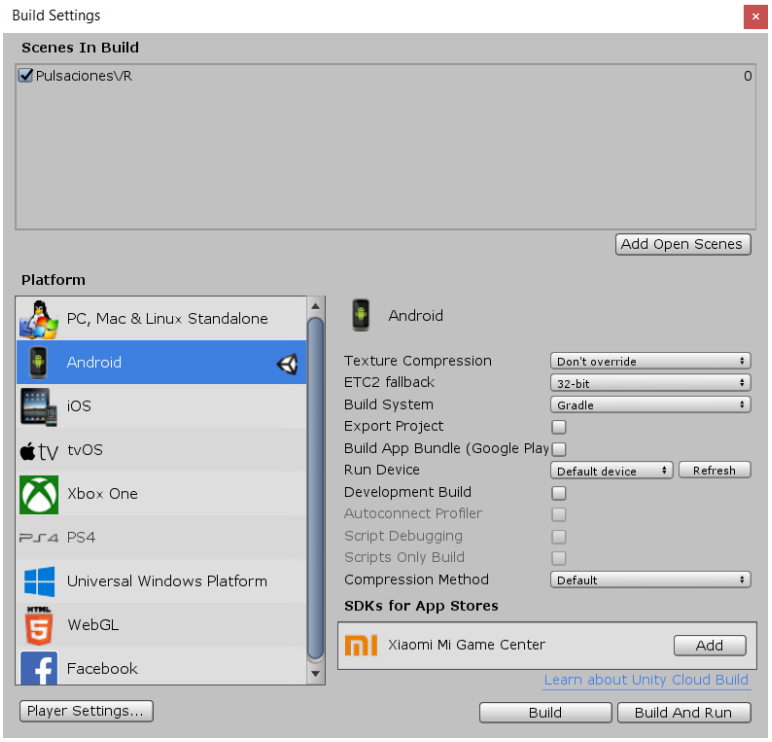
6.3 Implementación del juego en Unity. Componentes

Previamente hemos hablado sobre cada uno de los niveles que forman parte de "Pulsaciones VR", sin embargo, no hemos entrado en profundidad a explicar nada sobre la **configuración** que hay que utilizar para desarrollar juegos en realidad virtual en Unity, ni tampoco hemos hablado sobre los **componentes** que forman parte de la escena.

6.3.1 Configuración de Unity

Como el objetivo es desarrollar una aplicación ejecutada desde móviles Android debemos preparar el IDE de desarrollo para ello. Los pasos a seguir son los siguientes:

1. **Seleccionar como plataforma de desarrollo Android**
   Para poder escoger la plataforma sobre la que buscamos desarrollar un proyecto empleando Unity simplemente debemos de ir a la pestaña "***File***", seleccionar la opción "***Build Settings***" y ahí aparecerá una sección con las diferentes opciones a escoger. Es necesario este paso ya que por defecto Unity tiene marcada la opción para la implementación de aplicaciones para ordenador.



En el caso de no aparecer Android como alternativa es necesario instalar el SDK vinculado a esta plataforma para Unity. Los pasos a seguir para llevar a cabo este proceso vienen explicados en la API oficial del programa. [18]

A su vez, a la derecha de este apartado, en el caso de haber instalado el SDK y haber elegido Android, nos aparecerán otras características vinculadas a la ejecución del juego. Cabe destacar la opción "**Run Device**" ya que si tenemos conectado nuestro dispositivo móvil conectado al ordenador USB podemos ejecutarlo desde el mismo.

2. **Ajustes de juego**
   En la ventana mencionada en el punto anterior aparece un botón llamado "***Player Settings…***". Al pulsarlo nos aparecerá en la sección llamada **Inspector** una serie de ventanas en donde podemos especificar bastantes más preferencias.

   Para nuestro cometido únicamente debemos modificar los siguientes apartados:

   - Lo primero que debemos hacer es introducir un valor para los campos ***"Company Name"*** y "***Product Name***". Vinculado a esto en la subsección "***Other Settings***" en la opción "***Package Name"*** debemos de seleccionar como nombre del paquete la cadena formada por:

     com."***Company Name"."Product Name"***

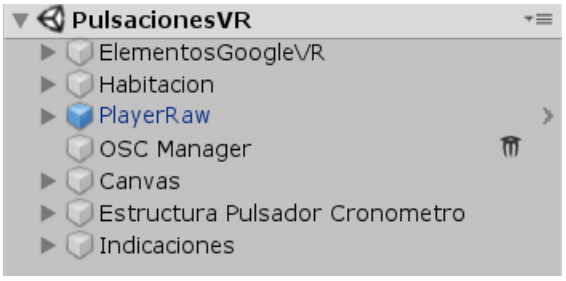
   - En ese mismo lugar en el apartado "**Minimum API Level**" debemos de elegir la opción **Android 4.4 Kitkat** como consecuencia de que es la versión del sistema operativo más pequeña que "***Cardboard***" puede soportar.

   - Finalmente, en la subsección ***"XR Settings"*** teniendo activado "***Virtual Reality Supported***" debemos añadir a "***Virtual Reality SDKs***" el que está relacionado con "***Cardboard***".

     Observación: Si quisiéramos también dar soporte para Oculus Rift, simplemente deberíamos añadir su SDK en este apartado.

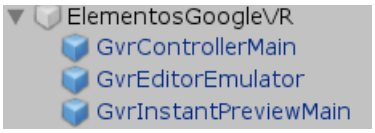
6.3.2 Componentes de la escena

Los elementos que forman parte del juego son los siguientes:



Elementos de Google VR [19][20]

Cuando empecé a desarrollar la aplicación tomé como base el ejemplo almacenado en el paquete para Unity de Google VR llamado **HelloVR**. En él se emplean multitud de componentes de los cuales he empleado solamente tres que son:



- **GvrControllerMain**
  Elemento fundamental para poder emplear en el juego la API asociada al controlador Daydream que empleamos en el mismo.

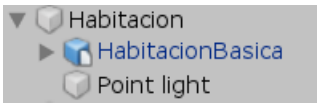
- **GvrEditorEmulator**
  Nos permite disponer de una vista previa del juego en el apartado "***Game***" del IDE de desarrollo en el que podemos simular el movimiento de la cabeza del jugador mediante el movimiento del ratón a la vez que mantenemos presionadas las teclas ALT o CTRL.

  Una vez que la aplicación se encuentra instalada en el dispositivo no tiene ninguna función.

- **GvrInstantPreviewMain**
  Es similar a GvrEditorEmulador, sin embargo, no es necesaria ninguna combinación de teclas, sino que únicamente necesita tener conectado el móvil al ordenador mediante USB o vía WIFI y todos los movimientos efectuados por éste serán captados en el emulador de Unity.

Habitación

Componente formado por la habitación creada mediante el programa SweetHome3D (explicado en el punto 4.2.3) y un foco que permite iluminar el interior de la misma.



Representación del jugador (Player Raw)

El siguiente elemento permite representar al jugador dentro del escenario. Está formado por dos componentes:

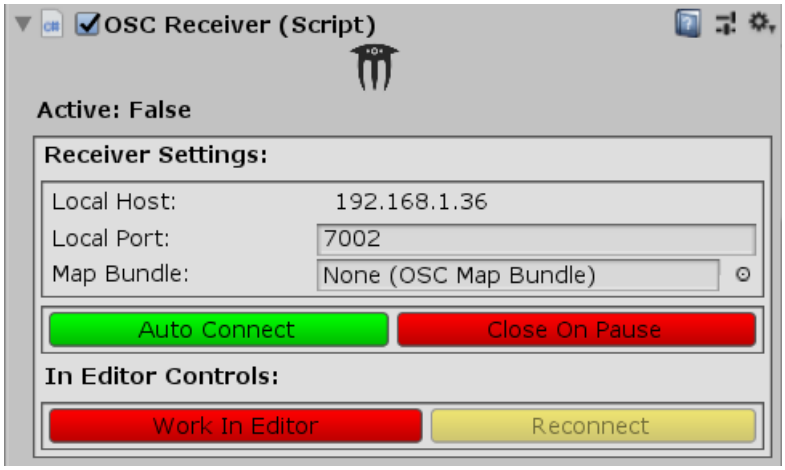
- **PlayerRaw**
  Se trata del componente que contiene la cámara principal del juego.

- **DefaultAvatar**
  Se trata del modelo en tres dimensiones que representa al jugador. Como se comentó en el punto en el que hablábamos sobre la **Asset Store,** el modelo utilizado no está realizado por mí, sino que emplee el avatar por defecto del paquete **Raw Mocap Data**.

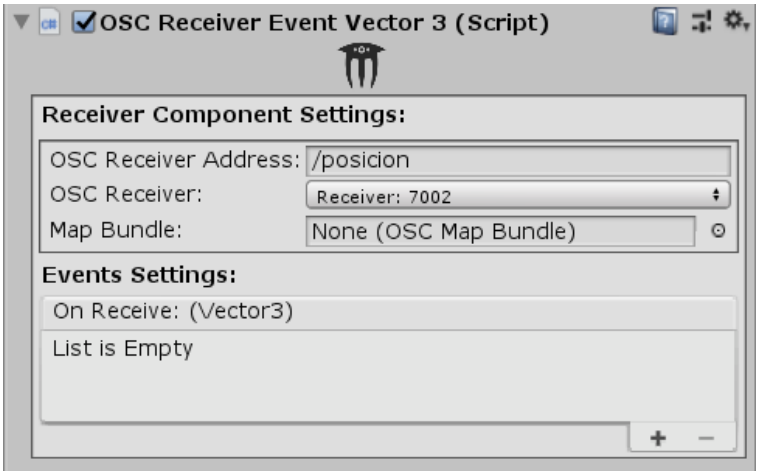
OSCManager

Se trata del componente más importante de la escena, ya que no solo se encarga de recibir cada uno de los paquetes OSC que son mandados desde la Raspberry Pi y los controladores de Arduino, sino que también gestiona la información que es recibida en cada uno de ellos. Presenta tres atributos que son los siguientes:

- **OSC Receiver**
  Elemento que forma parte del paquete de **OSCManager** que permite gestionar algunas características dentro de la recepción de los diferentes mensajes. Uno de los posibles cambios que nos permite realizar es el de establecer que puerto buscamos utilizar por defecto.



- **OSC Receiver Event Vector 3**
  Script asociado al receptor mencionado en el punto anterior que también forma parte del paquete de OSCManager. Sin embargo, ha sido modificado para que se encargue de desempaquetar los paquetes los cuales contienen la información asociada a la posición del usuario. Su representación en el IDE de Unity donde podemos elegir la dirección de los paquetes a tratar, así como que receptor escoger es la siguiente:

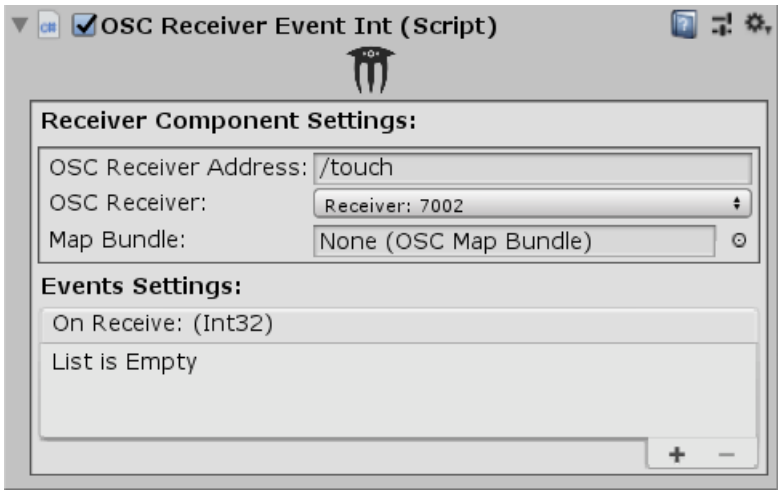


Por su parte, el contenido del código se muestra a continuación:

```
using UnityEngine;

using extOSC.Core.Events;
using System.Globalization;
using UnityEngine.UI;
using System;

namespace extOSC.Components.Events {
    [AddComponentMenu("extOSC/Components/Receiver/Vector3 Event")]
    public class OSCReceiverEventVector3 : OSCReceiverEvent<OSCEventVector3> {
        #region Protected Methods

        /*************************************************************
         * Variables
         *************************************************************/
        private Animator anim;                      // Variable que sirve para darle animación al personaje
                                                    // en el caso de que se reciba alguna modificación de la
                                                    // posición de este

        private GameObject Escenario;               // Variable que sirve para guardar el objeto que representa la habitación

        private GameObject player;                  // Variable que sirve para almacenar la representación del jugador

        private bool pos_detectada_inicial;         // Variable booleana que sirva para ver si se ha recibido un primer mensaje
                                                    // con información sobre la posición del jugador

        private float x_ant;                        // Variable x auxiliar que sirve para almacenar un valor de x previo que sirve para
                                                    // comparar con el nuevo valor de x con el fin de ver si el cambio a sudo suficiente
                                                    // como para activar la animación de movimiento o dejarlo en reposo.

        private float z_ant;                        // Lo mismo que en el caso de la variable x_ant solo que referente a la z


        public void Start() {
            // Localizamos los componentes asociados al jugador y al escenario
            player = GameObject.FindGameObjectWithTag("Player");
            Escenario = GameObject.FindGameObjectWithTag("Escenario");

            // Le damos inicialmente al modelo la posición de reposo
            anim = player.GetComponent<Animator>();
            anim.Play("Reposo", 0);
            pos_detectada_inicial = false;
        }


        protected override void Invoke(OSCMessage message) {

            // Como consecuencia de tener algún que otro problema con este tipo de información enviada y recopilada he tenido que recortar el string
            // recibido para obtener los datos
            const int tam_residual = 18;
            string recorte = message.ToString().Substring(28);
            string recorte_valores = recorte.Substring(0, recorte.Length - 1);
            char[] charSeparators = new char[] { ',' };
            string[] valores = recorte_valores.Split(charSeparators);
            // Con Substring(27) quitamos la primera parte del mensaje que recibimos
            int[] longitud = new int[] { valores.GetValue(0).ToString().Length, valores.GetValue(1).ToString().Length, valores.GetValue(2).ToString().Length };

            // Obtenemos los valores de las coordenadas x,y,z del mensaje
            float x = float.Parse(valores.GetValue(0).ToString().Substring(17, longitud[0] - tam_residual), CultureInfo.InvariantCulture.NumberFormat);
            float y = float.Parse(valores.GetValue(1).ToString().Substring(tam_residual, longitud[1] - 1 - tam_residual), CultureInfo.InvariantCulture.NumberFormat);
            float z = float.Parse(valores.GetValue(2).ToString().Substring(tam_residual, longitud[2] - 1 - tam_residual), CultureInfo.InvariantCulture.NumberFormat);

            // Tenemos los valores de x, y, z (En realidad solo nos interesa trabajar con los valores de x, z para poder desplazarnos)
            // Como en Unity la habitación estaba ubicada entre valores negativos y positivos:
            x = x - 250;
            z = z - 250;
```

```
68              /*****************************************************************************
69               *  Parte Animacion
70               *****************************************************************************/
71              if (!pos_detectada_inicial) {
72                  pos_detectada_inicial = true;
73                  x_ant = x;
74                  z_ant = z;
75
76                  anim.Play("Reposo",0);
77              }
78              else {
79                  if ((Math.Abs(x - x_ant) < 10) && ((Math.Abs(z - z_ant) < 10))) {
80                      anim.Play("Reposo",0);
81                  }
82                  else {
83                      anim.Play("Caminar",0);
84                  }
85                  x_ant = x;
86                  z_ant = z;
87              }
88
89              // Importante, la camara toma valores entre -250 y 250, mientras que el espacio del jugador va de -2.5 a 2.5, esta es la razón
90              // de porque dividimos los valores de x, y, z entre 100.
91              x = x / 100;
92              y = y / 100;
93              z = z / 100;
94              player.transform.position = new Vector3(-x, player.transform.position.y + y, z);
95          }
96
97          #endregion
98      }
99  }
```

- **OSC Receiver Event Int**
  Script vinculado a la gestión de aquellos paquetes que se obtienen a la hora de presionar cualquiera de los pulsadores. Su representación en el IDE de Unity únicamente varía con respecto al anterior en la dirección empleada ya que en vez de "**/posición"** se utiliza "**/touch":**



Con respecto al código asociado al script, este contiene toda la gestión necesaria para el correcto funcionamiento del juego en todos sus niveles.

```
using UnityEngine;

using extOSC.Core.Events;
using UnityEngine.UI;
using System.Collections;

namespace extOSC.Components.Events
{
    [AddComponentMenu("extOSC/Components/Receiver/Integer Event")]
    public class OSCReceiverEventInt : OSCReceiverEvent<OSCEventInt>{

        // Indicaciones
        private GameObject sonidoNivel1;
        private GameObject sonidoNivel2;
        private GameObject sonidoNivel3;
        private GameObject sonidoNivelFinal;
        private GameObject sonidoFin;
        private GameObject sonido30Segundos;
        private GameObject sonido10Segundos;
        private GameObject sonidoNuevoJuego;
        private GameObject sonidoVictoria;

        //-----------------------------------------
        // Pulsadores
        private GameObject pulsador1;
        private GameObject pulsador2;
        private GameObject pulsador3;
        private GameObject pulsador4;
        private GameObject pulsador5;
        private GameObject pulsador6;


        // Cronometros pulsadores y sus textos
        private GameObject cronometro1;
        private GameObject cronometro2;
        private GameObject cronometro3;
        private GameObject cronometro4;
        private GameObject cronometro5;
        private GameObject cronometro6;
        private Text texto_cronometro1;
        private Text texto_cronometro2;
        private Text texto_cronometro3;
        private Text texto_cronometro4;
        private Text texto_cronometro5;
        private Text texto_cronometro6;
        private Text texto_cronometro_aux;

        // Creamos una variable booleana para ver si un pulsador ha sido presionado
        // o no. Este funciona para que, en el caso de que el cronometro del pulsador
        // llegue a 0, no cambie el contador de pulsaciones
        private bool ha_sido_pulsado;

        // Por otro lado, necesitamos variables asociadas a los segundos de los
        // cronometros de los pulsadores.
        private float segundos_cronometro;
        private float temporizador_cronometro;
        private float segundo_cronometro;
```

```
// Gestion Camara
private Camera camara;
private GameObject[] instantCamara;
private float field_of_view_normal;          // Campo de visión del juego normal
private float fiel_of_view_modificado;       // Campo de visión del juego duplicado
private int id_pulsador;


private bool finJuego;                       // Variable que sirve para analizar si el juego se ha acabado
private bool bloqueado;                      // Variable que sirve para bloquear el juego una vez se pierda
private bool victoria;                       // Variable que sirve para ver si se ha conseguido completar
                                             // todos los niveles del juego


// Indicaciones Visuales
private GameObject temporizador_go;
private GameObject nivel_go;
private GameObject pulsaciones_go;
private GameObject reinicio_go;
private Text texto_temporizador;
private Text texto_nivel;
private Text texto_pulsaciones;


// Otras
private float temporizador;                  // Variable que sirve para almacenar el número de segundos que
                                             // tiene el temporizador del nivel. Irá decrementando su valor

private float segundos;                      // Variable que se asigna a la variable temporizador. Es un valor
                                             // fijo que no es modificado

private float segundo;                       // Variable utilizada para contar un segundo exacto para restarselo
                                             // a la variable temporizador.

private int pulsaciones_necesarias;          // Variable que sirve para almacenar la cantidad de pulsaciones que son
                                             // necesarias para pasar de nivel. Su valor no se ve modificado

private int cuenta_atras_pulsaciones;        // Variable a la que se le asigna el valor de pulsaciones necesarias
                                             // pero que si que va siendo modificada.

private bool inicia_nivel;                   // Variable que sirve para comprobar si un nivel ha acabado y empieza
                                             // otro.

private int nivel;                           // Variable que sirve para almacenar el nivel en el que estamos

private int pulsador;                        // Variable que irá almacenando el número aleatorio generado para ver que
                                             // pulsador toca presionar


public void Start() {
    // Localizamos la cámara esto lo haremos para cambiar la percepción de la distancia
    camara = GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>();
    instantCamara = GameObject.FindGameObjectsWithTag("InstantCamara");
    field_of_view_normal = 60f;
    fiel_of_view_modificado = 120f;
    camara.fieldOfView = field_of_view_normal;
    foreach (GameObject ic in instantCamara) {
        ic.GetComponent<Camera>().fieldOfView = field_of_view_normal;
    }
```

```
// En primer lugar, identificamos todos los pulsadores
pulsador1 = GameObject.FindGameObjectWithTag("Pulsador1");
pulsador2 = GameObject.FindGameObjectWithTag("Pulsador2");
pulsador3 = GameObject.FindGameObjectWithTag("Pulsador3");
pulsador4 = GameObject.FindGameObjectWithTag("Pulsador4");
pulsador5 = GameObject.FindGameObjectWithTag("Pulsador5");
pulsador6 = GameObject.FindGameObjectWithTag("Pulsador6");

// Por otro lado identificamos los cronómetros de los pulsadores, así como sus textos asociados
cronometro1 = GameObject.FindGameObjectWithTag("Cronometro1");
cronometro2 = GameObject.FindGameObjectWithTag("Cronometro2");
cronometro3 = GameObject.FindGameObjectWithTag("Cronometro3");
cronometro4 = GameObject.FindGameObjectWithTag("Cronometro4");
cronometro5 = GameObject.FindGameObjectWithTag("Cronometro5");
cronometro6 = GameObject.FindGameObjectWithTag("Cronometro6");
texto_cronometro1 = cronometro1.transform.GetChild(0).GetComponent<Text>();
texto_cronometro2 = cronometro2.transform.GetChild(0).GetComponent<Text>();
texto_cronometro3 = cronometro3.transform.GetChild(0).GetComponent<Text>();
texto_cronometro4 = cronometro4.transform.GetChild(0).GetComponent<Text>();
texto_cronometro5 = cronometro5.transform.GetChild(0).GetComponent<Text>();
texto_cronometro6 = cronometro6.transform.GetChild(0).GetComponent<Text>();

ha_sido_pulsado = false;
segundos_cronometro = 8;
temporizador = segundos_cronometro;


// Despues identificamos las indicaciones
sonidoNivel1 = GameObject.FindGameObjectWithTag("SonidoNivel1");
sonidoNivel2 = GameObject.FindGameObjectWithTag("SonidoNivel2");
sonidoNivel3 = GameObject.FindGameObjectWithTag("SonidoNivel3");
sonidoNivelFinal = GameObject.FindGameObjectWithTag("SonidoNivelFinal");
sonidoFin = GameObject.FindGameObjectWithTag("SonidoFin");
sonido10Segundos = GameObject.FindGameObjectWithTag("Sonido10Segundos");
sonido30Segundos = GameObject.FindGameObjectWithTag("Sonido30Segundos");
sonidoNuevoJuego = GameObject.FindGameObjectWithTag("NuevoJuego");
sonidoVictoria = GameObject.FindGameObjectWithTag("Victoria");
desactivarIndicaciones();

pulsador = 0;                // Como aun no hay ningún pulsador que hayamos presionado, su valor es 0
nivel = 1;
sonidoNivel1.SetActive(true);
finJuego = false;
bloqueado = false;
victoria = false;

// Gestionamos los segundos de una ronda
segundos = 60;
temporizador = segundos;
temporizador_go = GameObject.FindGameObjectWithTag("Temporizador");
texto_temporizador = temporizador_go.transform.GetChild(0).GetComponent<Text>();
texto_temporizador.text = "Tiempo restante \n" + temporizador.ToString() + " segundos";

// Gestionamos el contador de niveles
nivel_go = GameObject.FindGameObjectWithTag("Nivel");
texto_nivel = nivel_go.transform.GetChild(0).GetComponent<Text>();
texto_nivel.text = "Nivel\n"+ nivel +"\nPrueba Sin Tiempo";

// Gestionamos las pulsaciones necesarias
pulsaciones_necesarias = 3;
cuenta_atras_pulsaciones = pulsaciones_necesarias+1; // El "+1" es debido a que al empezar llamamos al método
                                                     // actualizarPulsador que modifica el valor de
                                                     // cuenta_atras_pulsaciones
```

```
    // Gestionamos el contador de niveles
    nivel_go = GameObject.FindGameObjectWithTag("Nivel");
    texto_nivel = nivel_go.transform.GetChild(0).GetComponent<Text>();
    texto_nivel.text = "Nivel\n"+ nivel +"\nPrueba Sin Tiempo";

    // Gestionamos las pulsaciones necesarias
    pulsaciones_necesarias = 3;
    cuenta_atras_pulsaciones = pulsaciones_necesarias+1; // El "+1" es debido a que al empezar llamamos al método
                                                         // actualizarPulsador que modifica el valor de
                                                         // cuenta_atras_pulsaciones

    pulsaciones_go = GameObject.FindGameObjectWithTag("Pulsaciones");
    texto_pulsaciones = pulsaciones_go.transform.GetChild(0).GetComponent<Text>();
    texto_pulsaciones.text = "Pulsaciones\nRestantes\n" + cuenta_atras_pulsaciones;

    // Gestionamos el mensaje de reinicio
    reinicio_go = GameObject.FindGameObjectWithTag("Reinicio");
    reinicio_go.SetActive(false);                                    // Lo ponemos a falso ya que este mensaje
                                                                     // aparecerá cuando se gane o pierda

    inicia_nivel = false;


    // Iniciamos el juego seleccionando al azar un pulsador a presionar
    actualizaPulsador();

}

public void Update() {
    // En el caso de que el juego no se haya bloqueado por
    // que aun no esté terminada la partida
    if (!bloqueado) {
        if (!finJuego) {
            gestionJuego();

        }
        else {
            // En el caso de completar todos los niveles
            // del juego
            if (victoria) {
                Debug.Log("Entro Victoria");
                StartCoroutine(hasGanado());
                actualizaPulsador();
                bloqueado = true;
            }
            // Si no consigues completarlos
            else {
                Debug.Log("Entro Derrota");
                StartCoroutine(hasPerdido());
                actualizaPulsador();
                bloqueado = true;
            }
        }
    }
}
```

```
/*********************************************************************
 *  Método Invoke
 *********************************************************************
 *  Método que es llamado cuando el juego recibe algún mensaje de
 *  tipo OSC
 *********************************************************************/
protected override void Invoke(OSCMessage message){
    int value;

    if (message.ToInt(out value)){
        // Analizamos el contenido de los mensajes
        if (onReceive != null) {
            // En el caso de que el valor recibido fuese 1
            if (value == 1) {
                // Si el pulsador que está activado es el 1, actualizamos el pulsador activo
                // En caso contrario no debe ocurrir nada
                if (pulsador1.activeSelf) {
                    if (bloqueado) {
                        bloqueado = false;
                        reiniciarJuego();
                    }
                    ha_sido_pulsado = true;
                    pulsador1.SetActive(false);
                    compruebaNivelActivo();
                    actualizaPulsador();

                }
            }

            // Si el pulsador que está activado es el 2, actualizamos el pulsador activo
            // En caso contrario no debe ocurrir nada
            if (value == 2) {
                if (pulsador2.activeSelf) {
                    if (bloqueado) {
                        bloqueado = false;
                        reiniciarJuego();
                    }
                    ha_sido_pulsado = true;
                    pulsador2.SetActive(false);
                    compruebaNivelActivo();
                    actualizaPulsador();
                }
            }
            // Si el pulsador que está activado es el 3, actualizamos el pulsador activo
            // En caso contrario no debe ocurrir nada
            if (value == 3) {
                if (pulsador3.activeSelf) {
                    if (bloqueado) {
                        bloqueado = false;
                        reiniciarJuego();
                    }
                    ha_sido_pulsado = true;
                    pulsador3.SetActive(false);
                    compruebaNivelActivo();
                    actualizaPulsador();
                }
            }
```

```
            // Si el pulsador que está activado es el 4, actualizamos el pulsador activo
            // En caso contrario no debe ocurrir nada
            if (value == 4) {
                if (pulsador4.activeSelf) {
                    if (bloqueado) {
                        bloqueado = false;
                        reiniciarJuego();
                    }
                    ha_sido_pulsado = true;
                    pulsador4.SetActive(false);
                    compruebaNivelActivo();
                    actualizaPulsador();
                }
            }
            // Si el pulsador que está activado es el 5, actualizamos el pulsador activo
            // En caso contrario no debe ocurrir nada
            if (value == 5) {
                if (pulsador5.activeSelf) {
                    if (bloqueado) {
                        bloqueado = false;
                        reiniciarJuego();
                    }
                    ha_sido_pulsado = true;
                    pulsador5.SetActive(false);
                    compruebaNivelActivo();
                    actualizaPulsador();
                }
            }
            // Si el pulsador que está activado es el 6, actualizamos el pulsador activo
            // En caso contrario no debe ocurrir nada
            if (value == 6) {
                if (pulsador6.activeSelf) {
                    if (bloqueado) {
                        bloqueado = false;
                        reiniciarJuego();
                    }
                    ha_sido_pulsado = true;
                    pulsador6.SetActive(false);
                    compruebaNivelActivo();
                    actualizaPulsador();
                }
            }
        }
    }

}
```

```
/*******************************************************************************
 * Función actualizaPulsador
 * Función que se encarga de realizar la gestión del cambio de pulsador
 * y del manejo del cambio de niveles entre los diferentes escenarios.
 *******************************************************************************/
private void actualizaPulsador() {
    // Buscamos obtener un numero entre el 1 al 6 (En la función Random.Range el valor máximo es excluido)
    id_pulsador = Random.Range(1, 7);

    // En caso de que no se haya pulsado aun ningún pulsador
    if (pulsador == 0) {
        pulsador = id_pulsador;
    }
    else {
        // Indicamos que un pulsador no pueda repetirse seguidamente
        while (id_pulsador == pulsador) {
            id_pulsador = Random.Range(1, 7);
        }
        // Actualizamos el valor del pulsador
        pulsador = id_pulsador;
    }

    // Actualizamos las variables del cronometro del pulsador
    temporizador_cronometro = segundos_cronometro;
    segundo_cronometro = 0;

    StartCoroutine(activaPulsador(pulsador));

    if (!finJuego) {
        if (ha_sido_pulsado) {
            ha_sido_pulsado = false;       // Actualizamos el valor de la variable
            cuenta_atras_pulsaciones--;    // Disminuímos el número de pulsaciones

            // Modificamos el texto informativo de la pared
            texto_pulsaciones.text = "Pulsaciones\nRestantes\n" + cuenta_atras_pulsaciones;

            // Realizamos la gestión del cambio de nivel
            if (cuenta_atras_pulsaciones == 0) {
                inicia_nivel = false;

                // En el caso de que nos encontremos en en alguno de los cuatro niveles jugables
                if (nivel < 5) {
                    // Aumentamos el nivel, lo indicamos en el texto de la pared y activamos el audio
                    nivel++;
                    // Actualizamos el número de pulsaciones necesarias para completarlo
                    cuenta_atras_pulsaciones = pulsaciones_necesarias;
                    desactivarIndicaciones();

                // En el caso de estar en niveles pares duplicamos el campo de visión
                if (nivel == 2 || nivel == 4) {
                    camara.fieldOfView = fiel_of_view_modificado;
                    foreach (GameObject ic in instantCamara) {
                        ic.GetComponent<Camera>().fieldOfView = fiel_of_view_modificado;
                    }
                }
                // En caso contrario, devolvemos este valor a la normalidad
                else {
                    camara.fieldOfView = field_of_view_normal;
                    foreach (GameObject ic in instantCamara) {
                        ic.GetComponent<Camera>().fieldOfView = field_of_view_normal;
                    }
                    desactivarTextoCronometros();
                }
                // Actualizamos el texto del numero de pulsaciones
                texto_pulsaciones.text = "Pulsaciones\nRestantes\n" + cuenta_atras_pulsaciones;
```

```
                // Comunicamos al jugador en que nivel se encuentra
                switch (nivel) {
                    case 1:
                        sonidoNivel1.SetActive(true);
                        break;

                    case 2:
                        sonidoNivel2.SetActive(true);
                        break;

                    case 3:
                        sonidoNivel3.SetActive(true);
                        texto_temporizador.text = "Tiempo restante \n" + temporizador.ToString() + " segundos";
                        break;

                    case 4:
                        sonidoNivelFinal.SetActive(true);
                        temporizador = segundos;
                        texto_temporizador.text = "Tiempo restante \n" + temporizador.ToString() + " segundos";
                        break;

                    default:
                        // Si el nivel es mayor que 4 quiere decir que ha ganado todos los niveles desarrollados
                        // por lo que habria ganado
                        //bloqueado = true;
                        victoria = true;
                        break;
                }

                // Gestionamos los temporizadores
                if (nivel != 5) {
                    if (nivel > 2) {
                        temporizador = segundos;

                        texto_nivel.text = "Nivel\n" + nivel;
                    }
                    else {
                        texto_nivel.text = "Nivel\n" + nivel + "\nPrueba Sin Tiempo";
                    }
                }
                else {
                    // En el caso de ganar el juego bloquear los mensajes de texto y mostramos el de reinicio
                    desactivarMensajes(false);
                    finJuego = true;
                }

            }
            // Si no se ha producido un cambio de nivel, simplemente actualizamos el pulsador
            else {
                StartCoroutine(activaPulsador(pulsador));
            }

        }

    }

}

/*************************************************************************************************
 * Función compruebaNivelActivo
 * Función que se encarga de estudiar si el nivel está activado o no. Sobre todo es útil para aquellos
 * niveles con cronómetro ya que hasta que no se active el nivel no empieza la cuenta atrás.
 *************************************************************************************************/
private void compruebaNivelActivo() {
    if (!inicia_nivel) {
        inicia_nivel = true;
    }
}
```

```
/*********************************************************************
 * Función activaPulsador
 * Función que se encarga de actualizar el pulsador a presionar.
 *********************************************************************/
IEnumerator activaPulsador(int pulsadorElegido) {
    // En principio desactivamos todos los pulsadores
    desactivarPulsadores();
    desactivarTextoCronometros();
    yield return new WaitForSeconds(1);

    switch (pulsadorElegido) {
        case 1:
            pulsador1.SetActive(true);
            if (nivel==2 || nivel == 4) {
                texto_cronometro1.gameObject.SetActive(true);
            }
            break;

        case 2:
            pulsador2.SetActive(true);
            if (nivel == 2 || nivel == 4) {
                texto_cronometro2.gameObject.SetActive(true);
            }
            break;

        case 3:
            pulsador3.SetActive(true);
            if (nivel == 2 || nivel == 4) {
                texto_cronometro3.gameObject.SetActive(true);
            }
            break;

        case 4:
            pulsador4.SetActive(true);
            if (nivel == 2 || nivel == 4) {
                texto_cronometro4.gameObject.SetActive(true);
            }
            break;

        case 5:
            pulsador5.SetActive(true);
            if (nivel == 2 || nivel == 4) {
                texto_cronometro5.gameObject.SetActive(true);
            }
            break;

        case 6:
            pulsador6.SetActive(true);
            if (nivel == 2 || nivel == 4) {
                texto_cronometro6.gameObject.SetActive(true);
            }
            break;

        default:
            Debug.Log("Numero aleatorio mal calculado");
            break;
    }
}
```

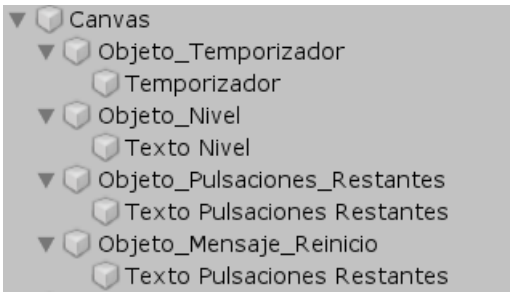
```
/*****************************************************************************************************
 * Función desactivarTextoCronometros
 * Función que se encarga de desactivar los textos de los cronometros de los pulsadores
 *****************************************************************************************************/
private void desactivarTextoCronometros() {
    texto_cronometro1.gameObject.SetActive(false);
    texto_cronometro2.gameObject.SetActive(false);
    texto_cronometro3.gameObject.SetActive(false);
    texto_cronometro4.gameObject.SetActive(false);
    texto_cronometro5.gameObject.SetActive(false);
    texto_cronometro6.gameObject.SetActive(false);
}

/*****************************************************************************************************
 * Función hasPerdido
 * Función que sirve para indicar al jugador que no ha podido completar el nivel en el tiempo
 * estipulado.
 *****************************************************************************************************/
IEnumerator hasPerdido() {
    desactivarMensajes(false);
    sonidoFin.SetActive(true);
    yield return new WaitForSeconds(2);
    sonidoNuevoJuego.SetActive(true);
}

/*****************************************************************************************************
 * Función hasGanado
 * Función que sirve para indicar al jugador que ha conseguido completar todos los niveles del
 * juego.
 *****************************************************************************************************/
IEnumerator hasGanado() {
    sonidoVictoria.SetActive(true);
    yield return new WaitForSeconds(4);
    sonidoNuevoJuego.SetActive(true);
}

/*****************************************************************************************
 * Función desactivarPulsadores
 * Función que permite desactivar todos los pulsadores de la escena.
 *****************************************************************************************/
private void desactivarPulsadores() {
    pulsador1.SetActive(false);
    pulsador2.SetActive(false);
    pulsador3.SetActive(false);
    pulsador4.SetActive(false);
    pulsador5.SetActive(false);
    pulsador6.SetActive(false);
}

/*****************************************************************************************
 * Función desactivarMensajes
 * Función que permite activar o desactivar los mensajes que son mostrados en una
 * de las paredes del juego.
 *****************************************************************************************/
private void desactivarMensajes(bool inicio) {
    if (inicio) {
        temporizador_go.SetActive(true);
        nivel_go.SetActive(true);
        pulsaciones_go.SetActive(true);
        reinicio_go.SetActive(false);
    }
    else {
        temporizador_go.SetActive(false);
        nivel_go.SetActive(false);
        pulsaciones_go.SetActive(false);
        reinicio_go.SetActive(true);
    }
}
```

```
/*****************************************************************************************
 * Método desactivarIndicaciones
 * Método que sirve para silenciar todas las indicaciones
 *****************************************************************************************/
private void desactivarIndicaciones() {
    sonido10Segundos.SetActive(false);
    sonido30Segundos.SetActive(false);
    sonidoFin.SetActive(false);
    sonidoNivel1.SetActive(false);
    sonidoNivel2.SetActive(false);
    sonidoNivel3.SetActive(false);
    sonidoNivelFinal.SetActive(false);
    sonidoVictoria.SetActive(false);
    sonidoNuevoJuego.SetActive(false);
}

/*****************************************************************************************
 * Método gestiónJuego
 * Método que se encarga de gestionar el tiempo en los niveles del juego
 *****************************************************************************************/
private void gestionJuego() {
    // En los dos primeros niveles no nos interesa tener tiempo ya que son de prueba
    // por lo que desactivamos el temporizador
    if (nivel==1 || nivel == 2) {
        texto_temporizador.text = "Fase de prueba\nEl temporizador no está activado";
    }
    else {
        // En caso de ya haber superado los niveles de prueba inicializamos la gestión
        // del temporizador. Hasta que uno de los pulsadores no sea presionado no
        // empezará la cuenta atrás.
        if (inicia_nivel) {
            if (temporizador > 0) {
                // Calculamos un segundo para después restarselo al total
                segundo = segundo + Time.deltaTime;
                if (segundo >= 1) {
                    segundo = 1;
                    temporizador = temporizador - segundo;
                    texto_temporizador.text = "Tiempo restante \n" + temporizador.ToString() + " segundos";
                    segundo = 0;

                    // En caso de que el numero de segundos que quedan sean 30 se lo indicamos mediante un
                    // comentario
                    if (temporizador == 30) {
                        desactivarIndicaciones();
                        sonido30Segundos.SetActive(true);
                    }
                    // Lo mismo hacemos cuando únicamente queden 10 segundos
                    else if (temporizador == 10) {
                        desactivarIndicaciones();
                        sonido10Segundos.SetActive(true);
                    }
                }
            }
            // Si el tiempo ha llegado a 0, el jugador ha perdido
            else if (temporizador <= 0) {
            // Sería necesario activar algún tipo de audio indicando que has perdido y modificar la pared
            // Diciendo que se ha perdido
                desactivarPulsadores();
                desactivarIndicaciones();
                desactivarTextoCronometros();
                nivel = 0;
                finJuego = true;
                victoria = false;
            }
        }
    }
```

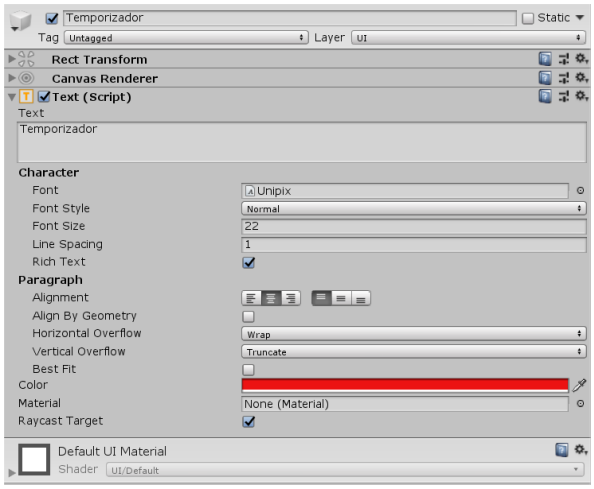
```
    // Por otro lado, tenemos que gestionar el tiempo de los cronómetros de los pulsadores
    if (nivel == 2 || nivel == 4) {
        if (temporizador_cronometro > 0) {
            segundo_cronometro = segundo_cronometro + Time.deltaTime;
            if (segundo_cronometro >= 1) {
                segundo_cronometro = 1;
                temporizador_cronometro = temporizador_cronometro - segundo_cronometro;

                // Obtenemos el texto del cronometro del pulsador
                actualizaTextoCronometroPulsadorActivo();
                segundo_cronometro = 0;

                if (temporizador_cronometro==0) {
                    actualizaPulsador();
                }
            }
        }
    }
}

/*****************************************************************************************************
 * Método actualizaTextoCronometroPulsadorActivo
 * Método que sirve para mostrar el temporizador asociado a cada uno de los pulsadores
 *****************************************************************************************************/
private void actualizaTextoCronometroPulsadorActivo() {
    switch (pulsador) {
        case 1:
            texto_cronometro1.text = temporizador_cronometro.ToString();
            break;
        case 2:
            texto_cronometro2.text = temporizador_cronometro.ToString();
            break;
        case 3:
            texto_cronometro3.text = temporizador_cronometro.ToString();
            break;
        case 4:
            texto_cronometro4.text = temporizador_cronometro.ToString();
            break;
        case 5:
            texto_cronometro5.text = temporizador_cronometro.ToString();
            break;
        case 6:
            texto_cronometro6.text = temporizador_cronometro.ToString();
            break;
    }
}

/*****************************************************************************************************
 * Método reiniciarJuego
 * Método que sirve para establecer todas los valores iniciales del juego
 *****************************************************************************************************/
private void reiniciarJuego() {
    // Reiniciamos todos los valores
    pulsador = 0;
    nivel = 1;
    cuenta_atras_pulsaciones = pulsaciones_necesarias + 1;
    sonidoNivel1.SetActive(true);
    finJuego = false;
    bloqueado = false;
    victoria = false;
    inicia_nivel = false;

    // Modificamos los mensajes de la pantalla
    desactivarMensajes(true);
    texto_temporizador.text = "Fase de prueba\nEl temporizador no está activado";
    texto_pulsaciones.text = "Pulsaciones\nRestantes\n" + cuenta_atras_pulsaciones;
    texto_nivel.text = "Nivel\n" + nivel + "\nPrueba Sin Tiempo";
}

#endregion
```

Indicaciones visuales del juego

Dentro del componente **Canvas** podemos encontrar los objetos del juego que contienen los textos que son mostrados en una de las paredes de la habitación. Aunque como podemos ver en una de las imágenes que se adjuntan en este apartado que el objeto que contiene el atributo de tipo texto tiene ya un valor fijado desde el IDE de desarrollo, este se verá modificado a medida que avancemos en el juego, tal y como aparece en el código escrito previamente relacionado con los pulsadores.



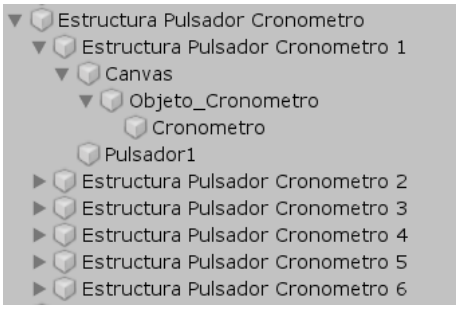
Estructura con los objetos y los textos que representan las indicaciones visuales



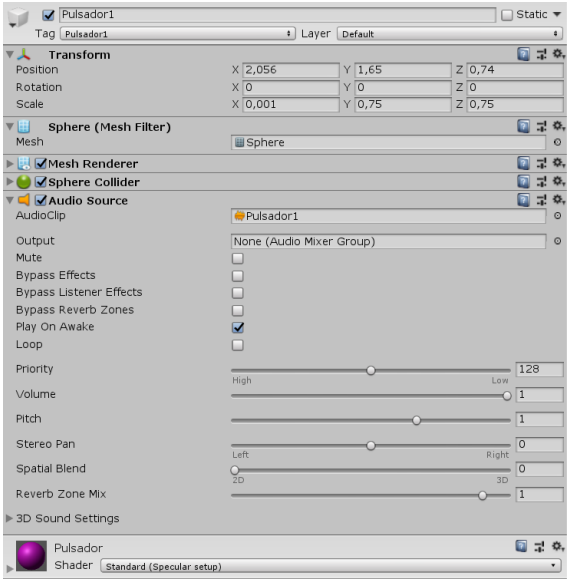
Ejemplo de definición del atributo Text

Estructura Pulsador-Cronómetro

Este elemento sirve para almacenar todos aquellos objetos formados por la unión de cada uno de los modelos que representan a cada pulsador y su respectivo cronómetro. Cada uno de éstos presenta un atributo de tipo *Audio Source* en el cual se le es asignado un sonido con una cierta frecuencia que hace a cada botón más identificable.



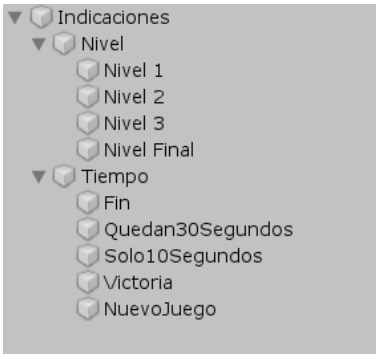
Esquema en el que se muestran todas las estructuras conformadas por un pulsador y su respectivo cronómetro



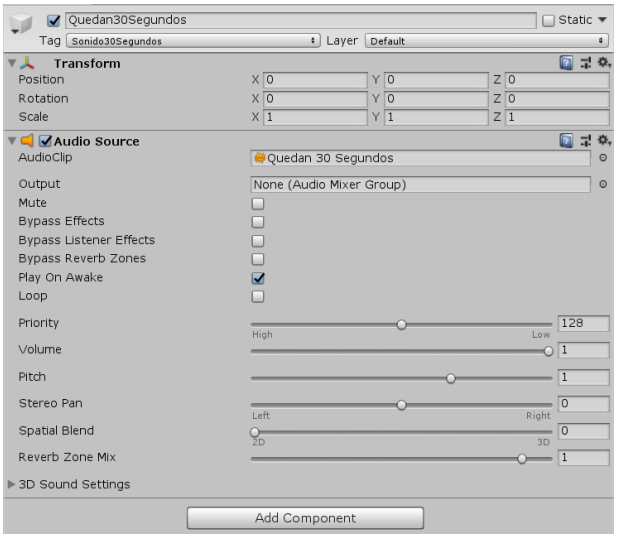
Ejemplo de asignación de sonido para uno de los pulsadores

Indicaciones auditivas del juego

Finalmente, el último "assert" por explicar es el que se encarga de almacena todos los objetos que son empleados de activar los mensajes que el usuario irá escuchando a medida que avance dentro del juego. Cada uno de ellos presenta un atributo de tipo *Audio Source* donde se le es asignado a cada indicación su sonido correspondiente.



Conjunto de objetos que forman parte del objeto Indicaciones



Ejemplo de asignación del sonido "Quedan 30 segundos"

6.3.3 Problemas encontrados durante el desarrollo

Principalmente caben destacar dos problemas importantes:

- **Orientación dentro del juego**
  Tanto los elementos empleados pertenecientes al paquete de Google VR, como la reproducción del juego mediante la plataforma de realidad virtual Daydream nos permiten emular con mucha precisión los movimientos que realizamos con la cabeza.

  Sin embargo, al iniciar la aplicación siempre se obtiene la misma imagen del juego (lugar donde la cámara está localizada por defecto). Esto supone un problema importante ya que en función de si inicializas la aplicación mirando a cualquier punto con el sentido incorrecto, provocará fallos de interpretación dentro del juego, haciendo por ejemplo que, si avanzas en la vida real, en el juego retrocedas.

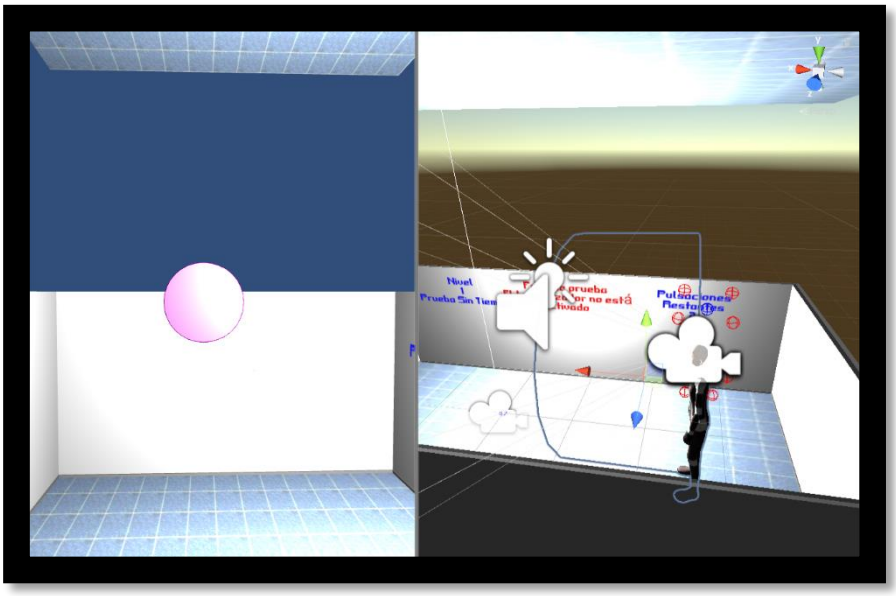
  Por lo que es imprescindible para su correcto funcionamiento del mismo fijar con que orientación se debe inicializar.

- **Cambio de *Field of View* (Campo de visión de la cámara)**

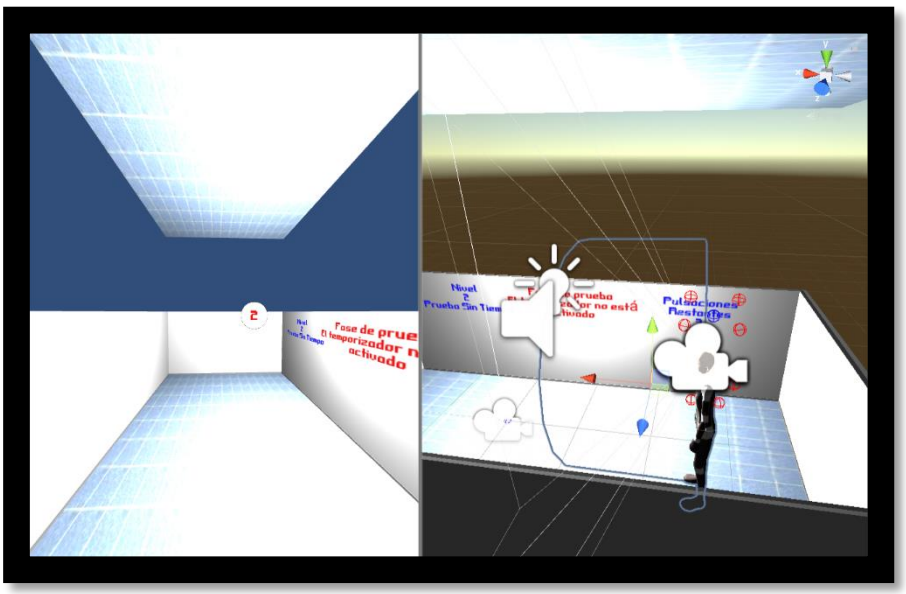
  Cuando originalmente pensamos en la idea de desarrollar diferentes niveles que el jugador tuviese que superar, en los niveles pares teníamos un cambio diferente con respecto a los impares que el que finalmente se ha implementado con el sistema de cronómetros incorporados a los pulsadores. El planteamiento original consistía en variar el campo de visión de la cámara para que a la hora de desplazarnos en el entorno nos pareciera que el camino a recorrer fuera más largo pero los pasos mucho más amplios y viceversa.

  Comparativa

  a) Visión con *Field of view* normal



  b) Visión con *Field of view* duplicado



Para poder hacer esto, simplemente es necesario cambiar desde código el valor de la variable "fieldOfView" asociado al objeto vinculado de la cámara.

No obstante, esto únicamente se ha conseguido realizar dentro del emulador del juego proporcionada por Unity. Tal y como hemos diseñado el juego con una única escena resulta imposible realizar este cambio en tiempo de ejecución.

Como puede indicarse en la documentación de Unity referente al entendimiento de la cámara, el usuario no puede cambiar el campo de visión durante el tiempo de ejecución. Esto se ha limitado debido a que hacer estos repentinos cambios en el tema de la realidad virtual puede inducir al mareo. [21]

Sin embargo, si que se podría llegar a simular este cambio en el campo de visión no cambiando dicho valor, sino alterando las escalas del entorno. Es decir, si multiplicamos la escala de la habitación, haciendo que la escala del objeto que representa a la cámara sea más pequeña, sí que podemos conseguir en cierta medida este objetivo. No obstante, por falta de tiempo no he podido llegar a implementar esta idea dentro del juego.

7. VARIACIONES Y EVALUACIÓN

7.1 Variaciones

Como he comentado en el punto en el que hablo sobre el estudio del arte, otro de los objetivos que nos marcamos fue la de exportar el proyecto ha Oculus Rift. El fin era la de comprobar la diferencia de rendimiento de "Pulsaciones VR" corriendo en estas plataformas.

Para ver esto decidimos grabar la ejecución del juego en la misma para tratar de ver con claridad dichas disimilitudes.

Android: https://drive.google.com/open?id=1fsLY12yOG7LZbH9BsS9PPNK_Ca8lxAVR

Oculus: https://drive.google.com/open?id=19BApTroz3V6JLL2VgVGkagGo57M_bRcS

Lo más apreciable a simple vista es la diferencia de *framerate* entre una y otra versión. Esto es como consecuencia de que en la versión de Android se van sacando fotogramas que a la vez que son tomadas están siendo analizadas para ver los cambios que se van produciendo entre cada una de ellas para detectar si hay o no movimiento.

Por otro lado, se aprecia una mayor inseguridad a la hora de moverte o localizar los pulsadores en este caso, como consecuencia de que el software detector de cambios en las imágenes no es del todo precisa. Puede darse el caso de que tome la sombra del jugador como parte del cambio situando al mismo a una pequeña distancia de donde se encuentra en realidad.

Este problema podría resolverse alterando los niveles de los parámetros que son pasados a las funciones de OpenCV vinculadas al tratamiento de imágenes. Sin embargo, esto haría que este proceso fuese más lento y haría inviable la jugabilidad.

Con Oculus no tenemos este problema, ya que previamente se ha debido establecer los límites de la zona de juego mediante la utilización de su sistema de sensores hace que tanto el cálculo de la posición como la fluidez este bastante más optimizada y precisa.

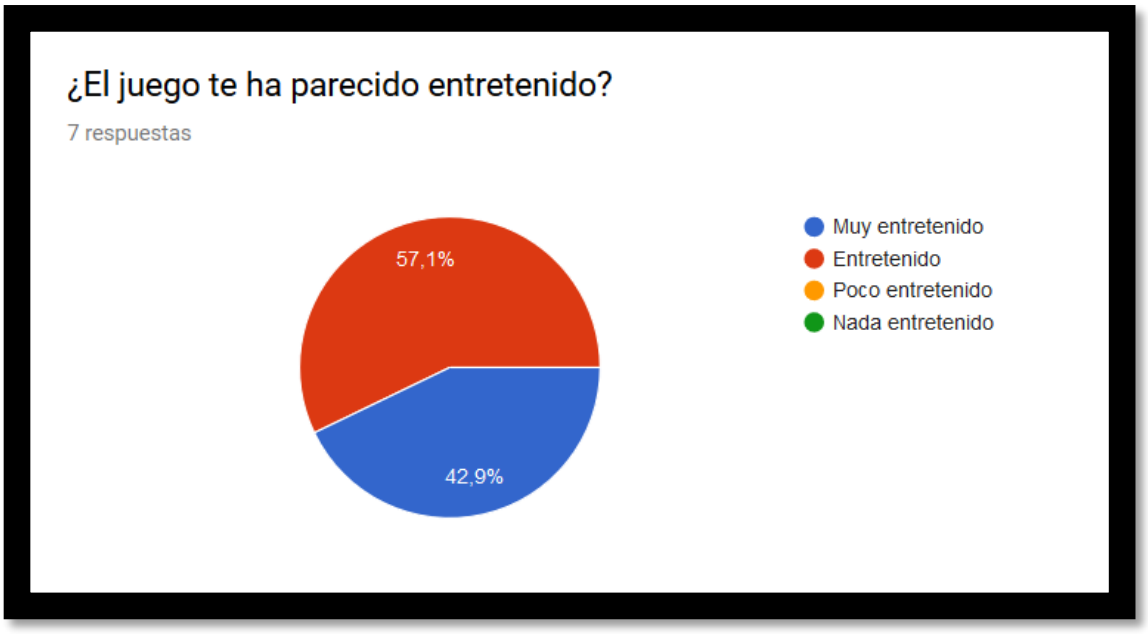
Otro aspecto a tener en cuenta es que con este último podemos agacharnos, dando la posibilidad de situar los pulsadores a diferentes alturas. En el caso de la ejecución tenga lugar en dispositivos móviles, aunque el juego este desarrollado en 3 dimensiones, de cara al estudio del posicionamiento no se ha tenido en cuenta la altura a la que se encuentre el jugador.

Para finalizar con este punto, cabe mencionar que en el caso de la ejecución en Oculus Rift no aparece representado el avatar del personaje debido a que su movimiento venía determinado por la posición que era calculada en la Raspberry, mientras que, en este caso, al calcularse el posicionamiento del usuario en función de la configuración del sistema guardián y el conjunto de sensores vinculados al dispositivo, se ha pensado en prescindir de éste.

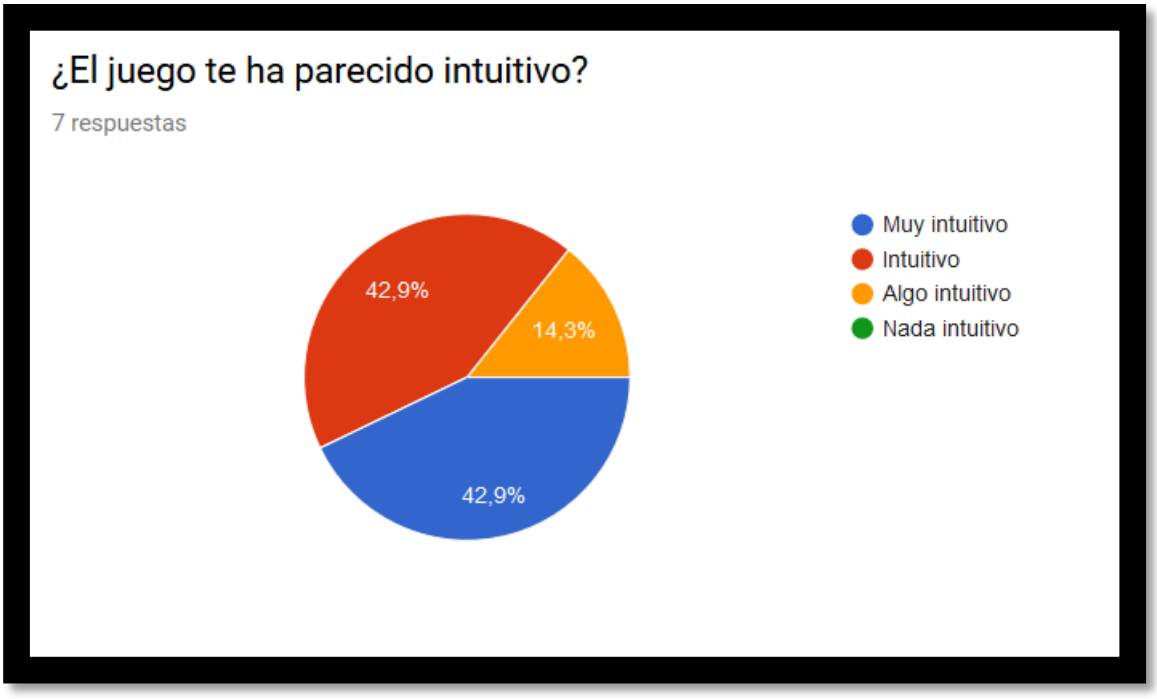
7.2 Test de usabilidad y posibles mejoras

Con el fin de ver los puntos positivos del juego, así como los aspectos a mejorar, decidí presentárselo a un grupo de personas a las cuales les hice llegar un cuestionario para conocer sus opiniones.

Las preguntas realizadas, así como las respuestas obtenidas son las siguientes:

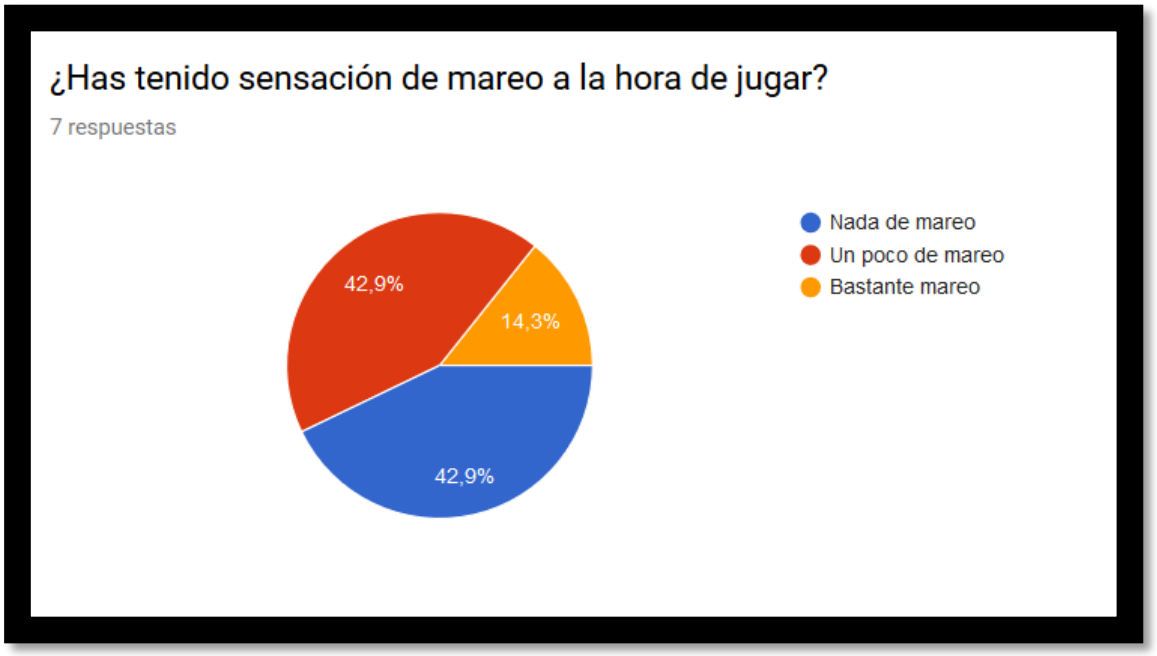


Como primera cuestión y un poco para romper el hielo, les preguntaba si se habían divertido a la hora de probar el juego. En general, los resultados han sido muy positivos no habiendo ninguna persona que lo considerara aburrido.



En segundo lugar, buscaba saber si con la forma en la que estaba diseñada la aplicación y con la explicación previa de la misma que les realicé antes de que los voluntarios la probaran sería suficiente para desenvolverse en la misma sin problemas.

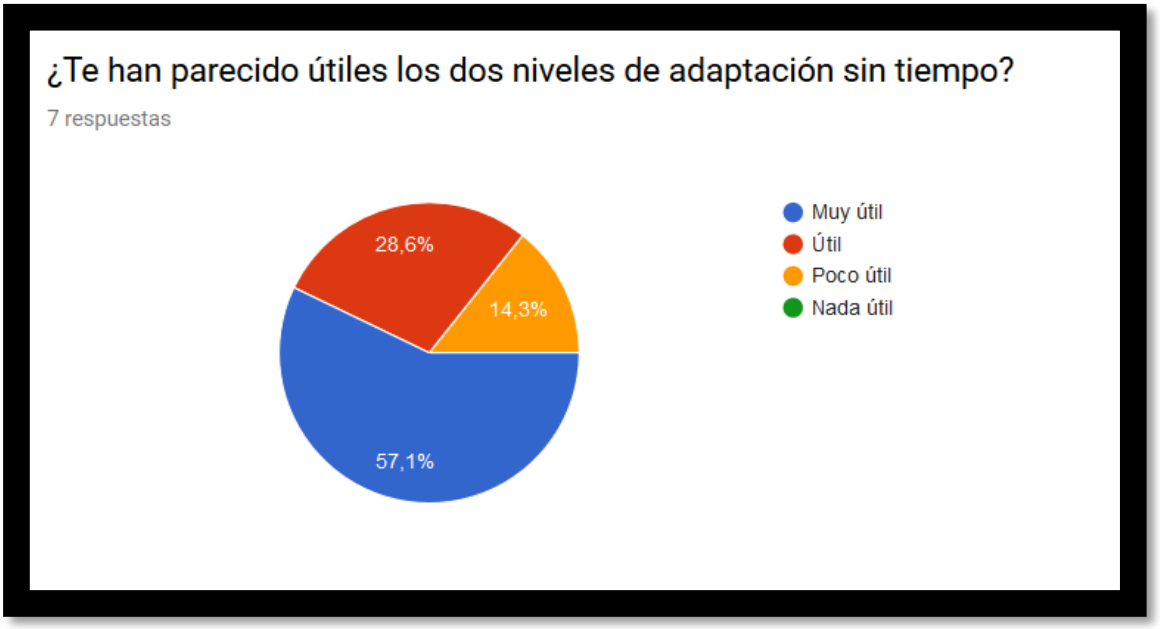
A pesar de que muchas de las respuestas dadas han sido positivas, me ha dado la sensación de que igual si que sea necesario algún tipo de pantalla donde se explique que hay que hacer en el juego acompañado de algún pequeño tutorial para situar al jugador de mejor manera.



Es sabido que estas nuevas tecnologías orientadas a la Realidad Virtual pueden hacer que algunas personas puedan llegar a sentirse mareadas tras utilizar algún software desarrollado para esta temática. Como hemos explicado en el documento Unity impidió el poder cambiar el campo de visión preocupados por este motivo.

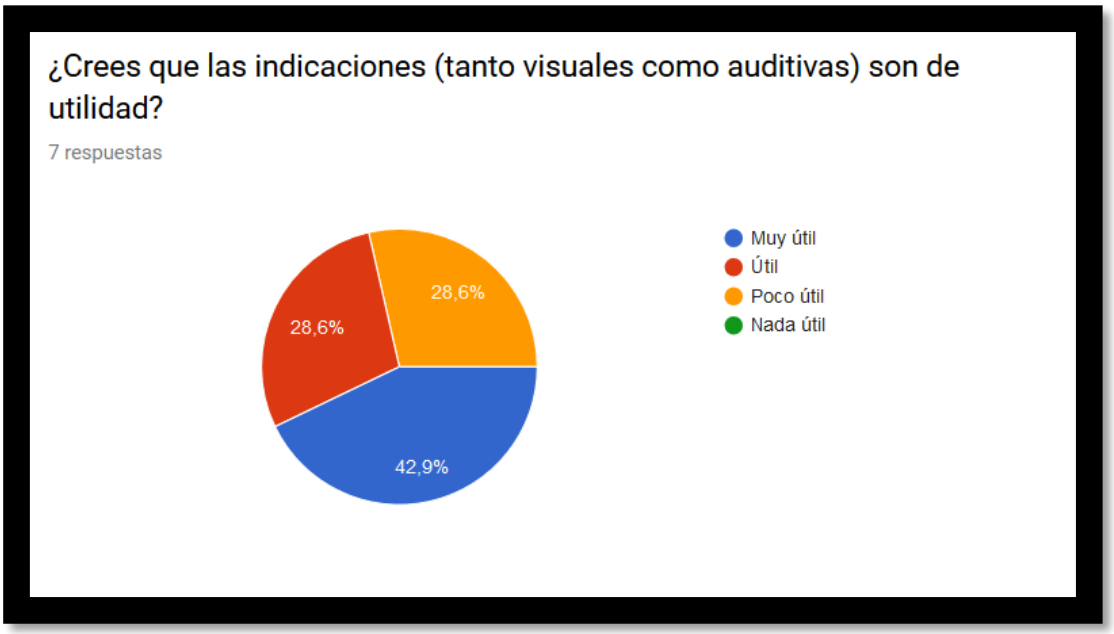
Además de tratarse de un juego basado en esta idea, hay que tener en cuenta que tenía un componente añadido de esfuerzo físico que podía incrementar este problema y esto se ha visto reflejado en la encuesta realizada. Más de la mitad de los voluntarios han tenido sensación de mareo, en algún caso más grave que en otros, a pesar de que el tiempo de cada una de las pruebas no era muy elevado (la duración media de cada una de ellas era inferior a 2 minutos).

Eso es algo a tener muy en cuenta, ya que quizás, si las actividades a desarrollar hubieran durado más tiempos, o si se habrían incluido más niveles, la salud de las personas podría haberse visto afectada en mayor medida.



Esta cuestión iba relacionada con la pregunta realizada previamente de si creían que el juego era intuitivo. Más de la mitad de los voluntarios han respondido que sí que era bueno primero adentrarse en el juego sin sentir la presión de poder escuchar el temido *Game Over*. Creíamos necesario introducir estos niveles ya que la gran mayoría no habían probado un juego con tan nivel de inmersión y que te da la posibilidad de desplazarte por un entorno.

La primera vez que lo pruebas puede originarse en ti una sensación de miedo a golpearte de frente con las paredes. Gracias a no tener ningún tipo de prisa, los usuarios de la aplicación pueden aprovecharlos para poder adaptarse al entorno, por lo que con estas respuestas ha quedado claro que ha sido una buena decisión introducirlos.



Con esta pregunta quería comprobar si las indicaciones ofrecidas en el momento de juego han sido o no de utilidad y se ha visto que es un aspecto que mejorar.

Vinculado a esta cuestión, les ofrecía un espacio en el formulario para que me comentaran posibles mejoras a realizar. Algunas de las respuestas han sido las siguientes:

"No me ha quedado claro el paso de un nivel a otro. Propongo algo más llamativo cuando se termina un nivel y empieza otro"

Si que es cierto que de un nivel a otro lo único apreciable rápidamente es el mensaje auditivo de cambio de nivel, pero que debido al sonido de los pulsadores pueda no percibirse del todo bien.

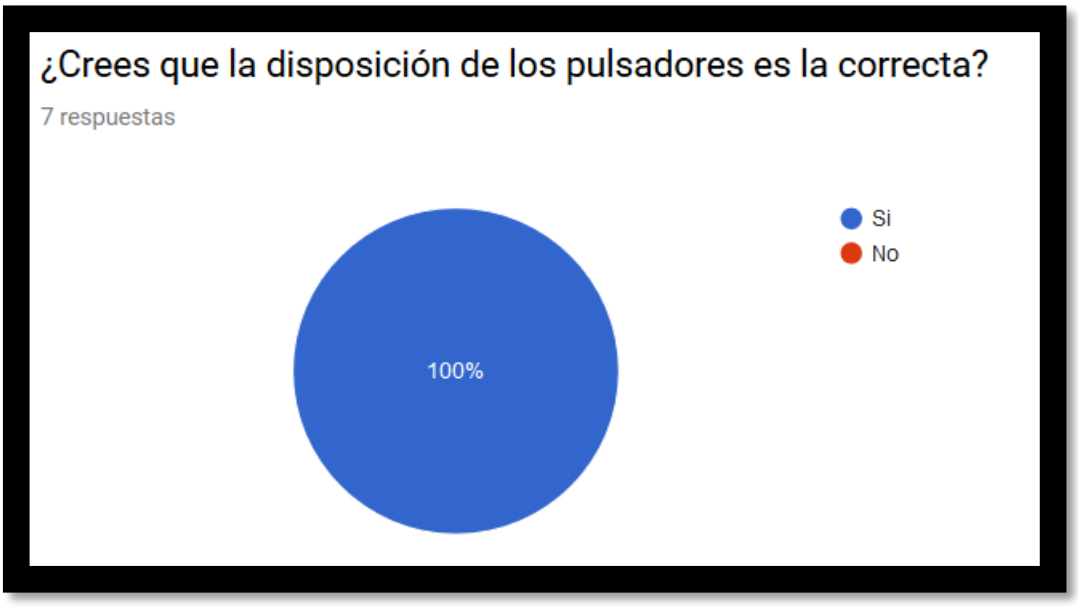
Una solución podría ser la de que en el visor se pueda contemplar un mensaje con el cambio de nivel conectado con la cámara del juego que independientemente de la posición o el movimiento del jugador se vea. Sin embargo, tampoco estaría mal cambiar el color de las paredes a medida que se va avanzando.

"Quizás la información que el tono de cada pulsador es más agudo o grave en función de la pared en la que se encuentra"

Para que su experiencia fuera más sencilla les comenté que la posición de los pulsadores estaba relacionada con la tonalidad del sonido. Era consciente de que a la hora de jugar esto no sería de gran ayuda, más que nada pensaba en indicar al jugador de que se había producido un cambio de pulsador.

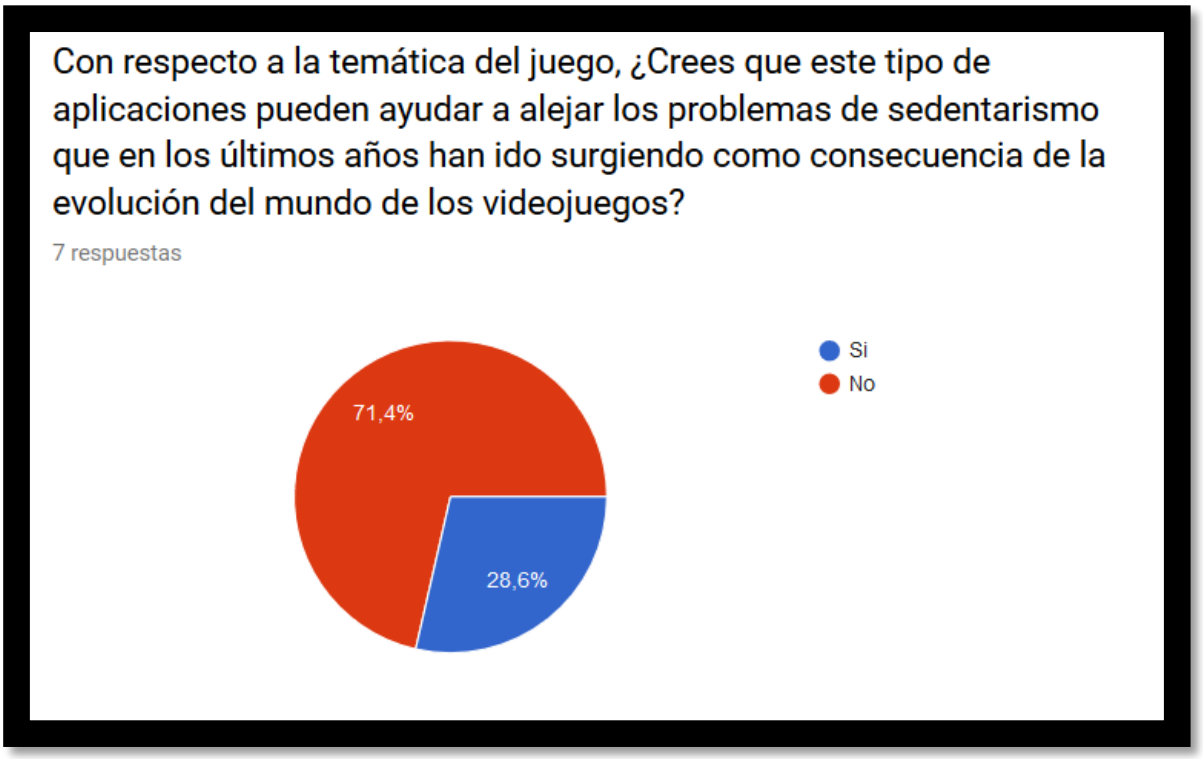
Quizás para facilitar el proceso de juego, en vez de que sonara un pitido se le indicara el identificador del pulsador recién activado. La información de que valor tiene cada pulsador sería revelada en el tutorial previamente comentado.

También buscaba conocer que les había parecido la adaptación del movimiento de la persona en el mundo virtual desarrollado. En general, las respuestas han sido muy positivas, aunque si que es cierto que este proceso puede tener algunos fallos, tanto en la detección de la posición por medio del sistema de cálculo de la misma en el caso de ejecutar la aplicación en dispositivos móviles, como por que los sensores no detecten correctamente al usuario como consecuencia de que algo lo imposibilite poniéndose en su trayectoria, en el caso de Oculus Rift.



Durante el planteamiento del juego dudé seriamente sobre cual sería la mejor posición de los pulsadores en la sala. Había pensado en ponerlos en el suelo, pero la placa Arduino correría mucho peligro de ser golpeada y estropeada. En el techo, si la altura fuera pequeña también podría situarse, esto hablando para el juego ejecutado desde Oculus, ya que para el cálculo de la posición mediante Raspberry Pi se necesita una altura elevada y sería inviable hacerlo.

Solo nos quedaba la opción de colocar pulsadores en las paredes, que como se ha comprobado ha sido una buena elección, pero si que se podría haber incluido otras mejoras, como la de utilizar en vez de dos, las cuatro paredes y la información sería mostrada activándola mediante el envío de mensajes OSC desde un dispositivo que actuara de mando, por ejemplo, un móvil o tablet.



La última pregunta va relacionada con el problema de que en los últimos años se ha vivido entre los jóvenes de que salgan menos a la calle a jugar por hacerlo desde casa. Al requerir algo de esfuerzo físico para completar todos los niveles pensaba que la respuesta sería unánime en que sí que podría ayudar a alejar este tipo de cuestión, pero las respuestas me han mostrado que estaba equivocado.

8. CONCLUSIÓN

A pesar de las diferencias mencionadas en el apartado anterior, como puede apreciarse en los videos adjuntos la experiencia de juego en ambos casos es satisfactoria, aunque mejorable. También es verdad que el código podría haber estado más optimizado en varias clases, sin embargo preferí realizarlo de esa manera al tener una fuerte dependencia del paquete de Unity encargado de la recepción de los mensajes.

Teniendo en cuenta las recomendaciones dadas por los voluntarios creo que el proyecto puede llegar a evolucionar bastante y a medida que las tecnologías mejoren se puede conseguir que problemas como el mareo puedan llegar a solventarse.

Dejando un poco al lado el juego en sí, me gustaría destacar todo lo que hay detrás del mismo. Ha sido muy gratificante poder desarrollar un trabajo final de grado con una temática que me apasiona.

Todo el tiempo invertido en "Pulsaciones VR" ha sido mucho más llevadero gracias a este motivo y sobre todo me ha permitido aprender una gran cantidad de conocimientos que de cara al futuro los tendré muy en cuenta.

Cuando me preguntaban en el colegio y en la propia universidad cual era mi objetivo en la vida que estuviese relacionado con mis estudios yo siempre respondía que consistía en desarrollar algo que hiciese felices a las personas que lo probasen. El día que los alumnos del máster vinieron a probar el juego, al ver sus sonrisas tras probarlo, tuve una pequeña dosis de ese sueño. Ya con eso estoy muy satisfecho del trabajo realizado a pesar de ser consciente de no ser algo perfecto.

Para finalizar, me gustaría dar las gracias ha todas las personas que han estado apoyándome en todo momento, a mi familia, a mis amigos, a toda la comunidad relacionada con la Universidad Pública de Navarra y en especial al profesor Oscar Ardaiz, el cual me proporcionó la oportunidad de afrontar este reto y que no dudó en ayudarme en momentos en los que podría sentirme atascado durante la realización del mismo.

9. BIBLIOGRAFÍA

[1] Tijmen van Willigen, "Touch Piano – Arduino capacitive sensor instrument", 2014 [Vídeo]
Disponible en:
https://www.youtube.com/watch?v=TMZs5zPzXVQ

[2] Abdualziz Alawshan, "Arduino-10: Fourth Experiment:Simon Says", 2016 [Vídeo]
Disponible en:
https://www.youtube.com/watch?v=XSfD6KNduVA

[3] Hacker Shack, "Smart Security Camera", 2017 [En línea]
Disponible en:
https://www.hackster.io/hackershack/smart-security-camera-90d7bd

[4] "VirtualSpace – Overloading Physical Space with Multiple Virtual Reality Users", 2018 [En línea]
Disponible en:
https://hpi.de/baudisch/projects/virtualspace.html

[5] Road to VR, "Oculus Touch Roomscale Tracking Test", 2016 [Vídeo]
Disponible en:
https://www.youtube.com/watch?v=abTo0JpjNyg

[6] Nathie, "EASTER EGG HUNT IN VIRTUAL REALITY | First Contact VR (Oculus Touch Gameplay)", 2017 [Vídeo]
Disponible en:
https://www.youtube.com/watch?v=Swx3nZrTtbc

[7] Adrian Rosebrock, "Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3", 2016 [En línea]. Disponible en:
https://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/

[8] "Installing operating system images" [En línea].
Disponible en:
https://www.raspberrypi.org/documentation/installation/installing-images/README.md

[9] "WEMOS D1 ESP8266 WIFI – Una placa cómoda para trabajar con ESP8266", 2018 [En línea].
Disponible en:
https://www.prometec.net/wemos-d1-esp8266-wifi/

[10] "Adafruit MPR121 12-Key Capacitive Touch Sensor Breakout Tutorial", [En línea]. Disponible en:
https://learn.adafruit.com/adafruit-mpr121-12-key-capacitive-touch-sensor-breakout-tutorial/wiring

[11] "Unity Documentation – Scripting API", 2019 [En línea].
Disponible en:
https://docs.unity3d.com/ScriptReference/

[12] "Make Human – Documentation: Index", 2018 [En línea].
Disponible en:
http://www.makehumancommunity.org/wiki/Documentation:Index

[13] V. Sigalkin, "GitHub-extOSC", 2017 [En línea]
Disponible en:
https://github.com/Iam1337/extOSC

[14] "Guía de Usuario de Sweet Home 3D", 2018 [En línea].
Disponible en:
http://www.sweethome3d.com/es/userGuide.jsp

[15] "OpenCV API Reference", 2019 [En línea].
Disponible en:
https://opencv.org/opencv-3-0/

[16] Adrian Rosebrock, "Basic motion detection and tracking with Python and OpenCV", 2015 [En línea]. Disponible en:
https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/

[17] Okay Dexter, "Motion detection using Python & OpenCV Contours", 2018 [Vídeo].
Disponible en:
https://www.youtube.com/watch?v=NrkEXsCxkhw

[18] "Android environment setup", 2018 [En línea].
Disponible en:
https://docs.unity3d.com/es/current/Manual/android-sdksetup.html

[19] "Releases Google VR for Unity", 2019 [En línea].
Disponible en:
https://github.com/googlevr/gvr-unity-sdk/releases

[20] "Quickstart for Google VR SDK for Unity with Android", 2019 [En línea].
Disponible en:
https://developers.google.com/vr/develop/unity/get-started

[21] "Understanding the camera" ***Unity Documentation – VR overview*****, 2019 [En línea].**
Disponible en:
https://docs.unity3d.com/Manual/VROverview.html#UnderstandingTheCamera